CHAPTER

Synthesis of clock and power/ground networks

13

Cheng-Kok Koh
Purdue University, West Lafayette, Indiana

Jitesh Jain
Purdue University, West Lafayette, Indiana

Stephen F. Cauley
Purdue University, West Lafayette, Indiana

ABOUT THIS CHAPTER

Clock distribution networks and power delivery systems are the two largest types of on-chip interconnect networks. They both play a crucial role in the correct operation of a circuit. A clock network delivers a synchronizing signal across the chip to coordinate the flow of data. A ***power/ground*** (P/G) supply network provides a reference voltage for determining the status of a transistor (on or off) and also the current for switching a transistor.

This chapter addresses many issues that affect the integrity of clock networks or P/G networks. We begin with a discussion of the timing, power, and robustness issues that one must consider when designing a clock network or P/G network. The chapter then examines the automated analysis, synthesis, and optimization of such large-scale interconnection networks in two sections, one for clock networks and the other one for power supply networks. For each section, we first cover the typical topologies encountered. After presenting the modeling and analysis techniques for such networks, the chapter describes algorithms to synthesize and optimize these networks.

13.1 INTRODUCTION

For a sequential circuit to operate correctly, the processing of data must occur in an orderly fashion. In a synchronous system, the order in which data are processed is coordinated by a clock signal. (To be more general, the

synchronization could be performed by a collection of globally distributed clock signals.) The clock signal, in the form of a periodic square wave that is globally distributed to control all sequential elements (flip-flops or latches), achieves synchronization of the circuit operation when all data are allowed to pass through the sequential elements simultaneously. A clock network is required to deliver the clock signal to all sequential elements. As the distributive nature of long interconnects becomes more pronounced because of technology scaling, the control of arrival times of the same clock edge at different sequential elements, which are scattered over the entire chip, becomes more difficult. If not properly controlled, the clock skew, defined as the difference in the clock signal delays to sequential elements, can adversely affect the performance of the systems and even cause erratic operations of the systems (*e.g.,* latching of an incorrect signal within a sequential element).

The power delivery system is in the form of two networks, one for providing the power supply voltage (V_{DD}) and one for providing the ground (GND or V_{SS}). We refer to them as the ***power/ground*** (P/G) network. A well-designed P/G network provides clean voltage references and adequate current for reliable and fast computation. Variations in V_{DD} and GND, commonly known as power supply noise, may result in logic failures or severe speed loss in the circuit. One reason for the fluctuations in V_{DD} and GND is that the P/G network spans the entire chip, because essentially all functional blocks (gates, memory cells, registers, etc.) on the chip have to be connected to the P/G pads. The presence of resistive (R) and inductive (L) parasitics in P/G pins and the P/G network leads to current-induced IR and $L \cdot di/dt$ voltage variations, respectively, when current (I or i) flows to facilitate the switching activities of transistors. Because low supply voltages are typically used in modern-day integrated circuits, the margin available to tolerate voltage variations is usually very stringent.

When considered independently, the design of a clock or P/G network already poses a formidable challenge because of stringent requirements. A well-designed clock or P/G network must also account for variations in device and interconnect parameters. What complicates matters even further is the interplay between the clock network and the P/G network. For example, in a zero-skew clock network, all sequential elements are triggered almost simultaneously. Because these elements draw current from the power network or sink current to the ground network almost simultaneously when the clock switches, the zero-skew design often leads to severe power supply noise, resulting in unacceptable degradation in performance and reliability. On the other hand, power supply noise affects the clock jitter (this refers to the time shift in the clock pulse, as well as the variations in the pulse width that arise from the time-varying operating conditions such as supply voltage), which, in turn, affects the arrival times of clock signal at different sequential elements.

We will describe in the next section many of the design considerations that one has to consider when implementing a clock network or a P/G network. These design considerations are captured either in the objective function for

which a synthesis algorithm for such networks has to optimize or as constraints that the synthesis algorithm has to satisfy. Synthesis algorithms for clock networks and P/G networks are topics that we cover in Sections 13.3 and 13.4, respectively.

13.2 DESIGN CONSIDERATIONS

Consider a simple synchronous circuit that uses positive edge-triggered *flip-flops* (FF's) as the sequential elements controlled by a clock signal that has even duty cycle (*i.e.,* a single-phase clocking scheme). We use $S = \{s_1, s_2, \ldots, s_n\}$ to denote the set of clock pins of flip-flops in the circuit, with s_i being the clock pin of flip-flop FF_i.

A pair of flip-flops is *sequentially adjacent* when only combinational logic exists between the two flip-flops. Let FF_i and FF_j be two sequentially adjacent flip-flops, with the output of FF_i being fed to the data input of FF_j through a combinational logic (see Figure 13.1). We say that FF_i is the launching flip-flop and FF_j is the capturing flip-flop.

13.2.1 Timing constraints

Suppose the clock signal arrives at all the flip-flops simultaneously. In Figure 13.2, the clock signal arrives at the 12 o'clock position.[1] Here, we assume an even duty clock signal. Hence, in the first half of the clock cycle the clock signal is high ("CLK = 1"). At the 6 o'clock position, the clock switches from high to low ("CLK = 0"). After a clock period of C_P, the clock signal is again at the 12 o'clock position.

The minimum allowable clock period C_P for the synchronous system must be long enough to accommodate the time it takes to propagate the signal through the

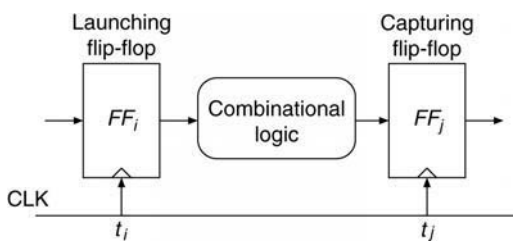


FIGURE 13.1

A simple sequential circuit.

[1]This representation was credited to Professor Mark Horowitz of Stanford University in the accompanying lecture materials of the book by [Rabaey 2003].

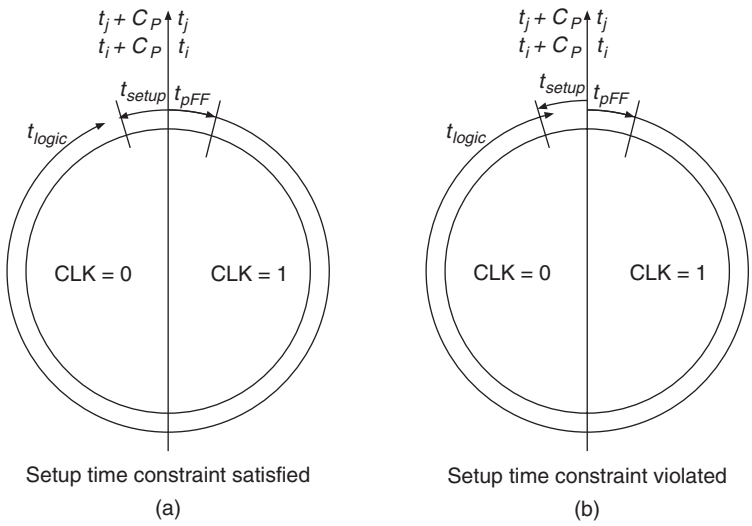


FIGURE 13.2

Setup time constraint.

launching flip-flop, denoted as t_{pFF}, and the time it takes for the signal to make its way through the combinational logic, denoted as t_{logic}. Moreover, the clock period must also account for the setup time, denoted as t_{setup}, which is the amount of time that the input to the capturing flip-flop has to stay valid before the next triggering clock edge arrives (see Figure 13.2a). This can be summarized by the following inequality, which is also known as the setup time constraint:

$$C_p \geq t_{pFF} + t_{logic} + t_{setup}$$

Of course, one should use the worst-case (or maximum) propagation delays for flip-flops and combinational logic and the maximum setup time requirement to determine the lowest operable clock period. Violation of the setup time constraint is also commonly referred to as a zero-clocking hazard, because the signal is not latched in properly by the capturing flip-flop (see Figure 13.2b).

The setup time constraint presents only one side of the story for the proper operation of a synchronous system. To ensure proper propagation of input signal through a flip-flop, the input must remain valid or hold steady for a short duration after the clock edge. That short duration is referred to as the hold time, denoted as t_{hold}. The hold time of a capturing flip-flop imposes an additional constraint on the total propagation delay of a signal through the launching flip-flop and the combinational logic as follows:

$$t_{pFF} + t_{logic} \geq t_{hold}$$

To accommodate the most extreme design corner, the preceding inequality, which is known as the hold time constraint, should be formed with the

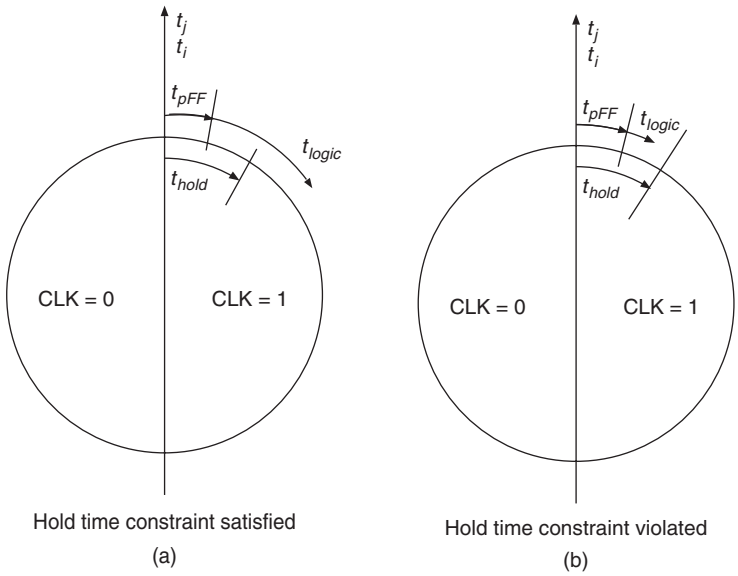


FIGURE 13.3

Hold time constraint.

maximum hold time of the capturing flip-flop and the minimum propagation delay of the launching flip-flop and the smallest propagation delay of the combinational logic (see Figure 13.3a). The violation of hold time constraint is commonly referred to as a double-clocking hazard, because two signals would have gone through the capturing flip-flop in a single clock cycle (see Figure 13.3b).

Remark: Clocking for Combinational Logic: The focus of this chapter is on clocking for sequential circuits (finite state machines or pipelined systems). Clocking is also an integral component of combinational logic: Dynamic circuits, such as Domino logic and NP CMOS logic (Zipper logic), require clock signals to synchronize the precharge phase and the evaluation phase of the circuits. It is important to realize that techniques presented in this chapter can be adapted to handle the synchronization of precharge-and-evaluate circuitry.

13.2.2 **Skew and jitter**

Because of the unbalanced delays in the clock distribution network, clock edges may arrive at clock pins s_i and s_j in Figure 13.1 at different times. Such *spatial variation* in the arrival times of a clock transition at two different locations on a chip is commonly known as the clock skew. Let t_i and t_j be the signal delays

from the clock source to s_i and s_j, respectively. Define the clock skew between s_i and s_j, denoted by $skew_{i,j}$, to be

$$skew_{i,j} = t_i - t_j$$

By definition, $skew_{i,j}$ stays constant from cycle to cycle [Rabaey 2003]. Figure 13.4 shows two different scenarios: (a) $skew_{i,j} > 0$ and (b) $skew_{i,j} < 0$. They, respectively, correspond to the clock signal arriving at s_j earlier than and later than at s_i.

At time t_i, the clock edge arrives at flip-flop FF_i. The input of FF_i takes $t_{pFF}^{max} + t_{logic}^{max}$, in the worst case, to propagate through the flip-flop and the combinational logic. The signal must have settled down for a duration of t_{setup}^{max} before the next clock edge arrives at time $t_j + C_P$ at FF_j, the capturing flip-flop, so that it can be properly latched in. That translates into the following more general setup time constraint:

$$t_i + t_{pFF}^{max} + t_{logic}^{max} + t_{setup}^{max} \leq t_j + C_p \tag{13.1}$$

Rewriting Equation (13.1), it is clear that a *positive* clock skew, as shown in Figure 13.4a, places a lower bound on the allowable clock period (or an upper bound on the operating frequency of the circuit $1/C_P$) as follows:

$$C_p \geq skew_{i,j} + t_{pFF}^{max} + t_{logic}^{max} + t_{setup}^{max} \tag{13.2}$$

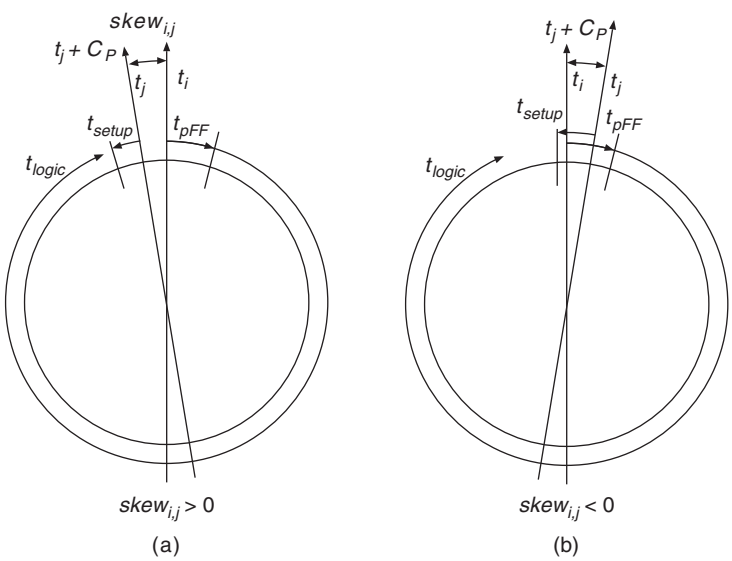


FIGURE 13.4

Setup time constraint in the presence of clock skew.

The terms in Equation (13.2) can also be rearranged into

$$skew_{i,j} \leq C_p - t_{pFF}^{\max} - t_{logic}^{\max} - t_{setup}^{\max} \tag{13.3}$$

which shows that for a given frequency, we must bound the skew from above.

Although a positive $skew_{i,j}$ always decreases the maximum attainable clock frequency, a negative clock skew (*i.e.,* $skew_{i,j} < 0$) actually increases the effective clock period, as shown in Figure 13.4b. In other words, we may have a combinational logic block between two sequentially adjacent flip-flops that has a propagation delay longer than the given clock period.

Example 13.1 Cycle Stealing or Useful Clock Skew. Consider a 3-stage pipeline as shown in Figure 13.5a. Let $t_{logic,CLi}$ denote the delay of combinational logic block *CLi*. In this example, we assume that $t_{logic,CL2} = 10$ ns, $t_{logic,CL1} = t_{logic,CL3} = 8.5$ ns. If the clock signal arrives at all clock pins at the same time (Figure 13.5a), it is obvious that the fastest frequency at which the circuit can operate is 100 MHz. However, that would mean that for

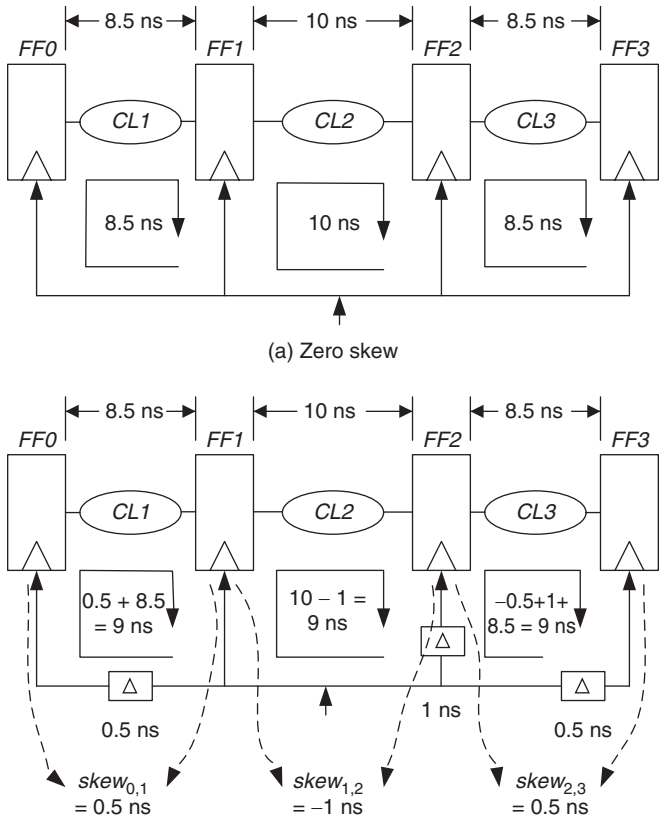


FIGURE 13.5

Cycle stealing.

the last 1.5 ns of the clock period, combinational logic blocks *CL*1 and *CL*3 are sitting idle. Although they maintain the correct values at *FF*1 and *FF*3 before the arrival of the next clock edge, they are not doing meaningful computation.

In the following, we will show how we can design the clock distribution network such that the system can run at 111 MHz (*i.e.,* a clock period of 9 ns). In this clock network, the clock signal arrives at *FF*0, *FF*1, *FF*2, and *FF*3 with the following skews: $skew_{0,1} = 0.5$ ns, $skew_{1,2} = -1$ ns, and $skew_{2,3} = 0.5$ ns. The effects of the skews are that *CL*1 and *CL*3 have exactly 8.5 ns for the computation, and *CL*2 has exactly 10 ns for the computation. In other words, *CL*2 steals 0.5 ns each from the clock periods for *CL*1 and *CL*3 so that the system can operate at a clock period that is smaller than the longest path delay in *CL*2.

However, when the skew is excessively negative, the signal may arrive so early that it races through the capturing flip-flop, resulting in double-clocking or a hold-time violation. Consider the example in Figure 13.1 again. At time t_i, the clock edge triggers flip-flop FF_i, and the input to FF_i propagates through the flip-flop and the combinational logic. Because we are considering the possibility of data racing with the clock signal, we use the shortest propagation delay through the flip-flop and the combinational logic, which is $t_{pFF}^{\min} + t_{logic}^{\min}$. In the worst case, the input signal at FF_j has to remain stable for $t_{hold}^{\max}$ after the clock edge of the same clock cycle arrives at t_j. Therefore, only after $t_j + t_{hold}^{\max}$ may the signal from FF_i erase the input at FF_j (see Figure 13.6). This constraint is expressed as follows:

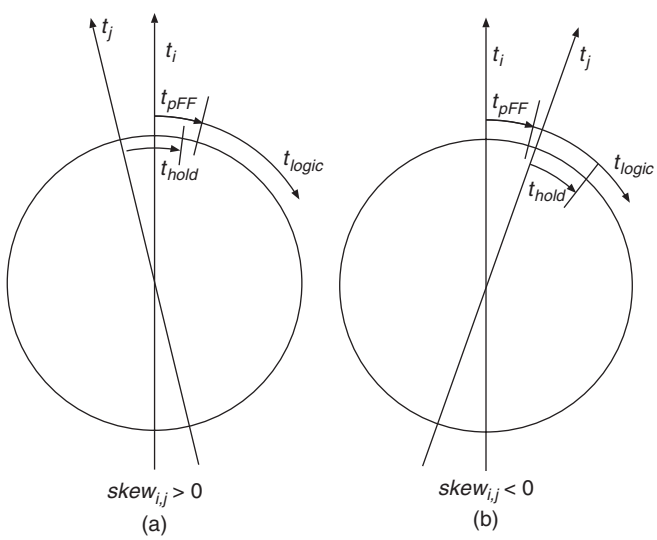


FIGURE 13.6

Hold time constraint in the presence of clock skew.

$$t_i + t_{pFF}^{\min} + t_{logic}^{\min} \geq t_j + t_{bold}^{\max}$$

which can be rewritten as a lower bound constraint on the skew:

$$skew_{i,j} \geq t_{bold}^{\max} - t_{pFF}^{\min} - t_{logic}^{\min} \tag{13.4}$$

As mentioned earlier, clock skew is typically caused by unbalanced delays in the clock distribution network. Such unbalanced delays can be attributed to uneven wire lengths along the clock paths, mismatches in the numbers of buffers along the clock paths, and differences in the clock load driven by clock buffers. Although it is possible to manipulate these differences and mismatches to create useful skew for performance enhancement, skew caused by variations in the manufacturing process cannot be eliminated or exploited easily. Oxide variations, dopant variations, and width and length variations of devices affect such device parameters as the threshold voltage and parasitic capacitance of clock drivers and buffers. Variations in the interconnect thickness, widths, and spacing, and variations in interlayer dielectric thickness affect the interconnect resistance and capacitance. Because of these variations in process parameters, the clock skew between two clock pins may be different in two different dies, although the skew remains constant in a die. The challenge is to design a clock distribution network that works equally well across different dies.

Although skew caused by static variations stays constant cycle to cycle, time-varying variations in the operating condition of a circuit cause *temporal variation* of the clock period at a given point on the chip [Rabaey 2003]. The clock edge at a flip-flop may sometimes arrive earlier and sometimes later with respect to an ideal reference clock, depending on the operating condition of the circuit, as shown in Figure 13.7. Such temporal clock-period variation is referred to as *clock jitter*. In particular, we call the worst-case deviation (absolute value) of the arrival time of a clock edge at a given location with respect to an ideal reference clock edge as *absolute jitter*. Let $t_i(n)$ and $t_i(n+1)$, respectively, refer to the arrival times of the nth and $n+1$st clock edges at clock pin s_i. The cycle-to-cycle jitter (absolute value) for s_i is defined to be $t_i^{jitter}(n) = |t_i(n+1) - t_i(n) - C_P|$. Hence, the worst-case cycle-to-cycle jitter is

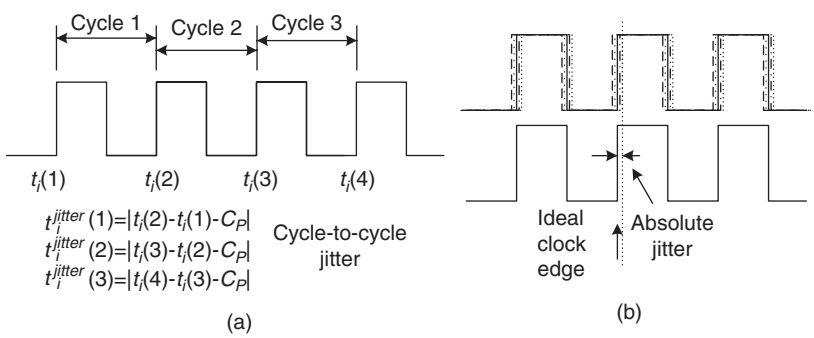


FIGURE 13.7

(a) Cycle-to-cycle jitter. (b) Absolute jitter.

twice the absolute jitter (*i.e.*, $max_n t_i^{jitter}(n) = 2T_i^{jitter}$), where T_i^{jitter} is the absolute jitter of s_i. The setup-time constraint and hold-time constraint should be updated accordingly:

$$skew_{i,j} \leq C_p - t_{pFF}^{\max} - t_{logic}^{\max} - t_{setup}^{\max} - (T_i^{jitter} + T_j^{jitter}) \tag{13.5}$$

$$skew_{i,j} \geq t_{hold}^{\max} - t_{pFF}^{\min} - t_{logic}^{\min} + (T_i^{jitter} + T_j^{jitter}) \tag{13.6}$$

In other words, the clock period may be shortened or lengthened by $T_i^{jitter} + T_j^{jitter}$.

A few main sources of environmental variations cause clock jitter: power supply noise, temperature gradients, substrate noise, and coupling of the clock network with adjacent signals. All these variations are caused by switching activities. The first three variations, in particular, affect the generation and propagation of clock signal, because device parameters are strong functions of these. The analog component of the clock generation circuitry and the clock buffers in the distribution network are both significant contributors to clock jitter. Switching activities, which vary from cycle to cycle, cause variations in interconnect couplings, gate capacitances, and the voltages of internal nodes of latches and flip-flops. As a consequence, the load presented to clock buffers varies from cycle to cycle, which causes jitter.

In the next section, we further elaborate on an important source of time varying variations, namely, the power supply noise.

13.2.3 *IR* drop and $L \cdot di/dt$ noise

Because of the non-zero resistance of wires connecting a transistor to a P/G pad, the transistor does not see a full V_{DD} when current flows from the P/G pad to the transistor. The voltage fluctuations, called *IR* drop, cause a degradation in drive strength. Until recently, *IR* drop has been defined mainly by considering the average current flowing through a power supply network [Lin 2001]. In a static *IR* drop analysis, we calculate the *IR* drop defined as such by performing a DC analysis of the power supply network that takes the average current as the input. Of late, designers are concerned that the voltage of a power supply network may not have sufficient time to recover because of the shrinking clock period as the clock frequency increases. There has been an increasing need for dynamic *IR* drop analysis, in which we have to consider the current waveform across multiple clock cycles [Lin 2001]. Therefore, the analysis of dynamic *IR* drop requires a transient analysis of the power supply network.

The P/G pins also have associated inductive parasitics. Moreover, wires in a P/G network typically have larger dimensions. Combined with the high operating frequencies of modern designs, the impedances of such wires are increasingly dominated by their inductive parasitics. Because of the time-varying nature of the current drawn from the power supply noise, inductance causes AC voltage fluctuations, called $L \cdot di/dt$ noise, in the supply lines. Large current spikes caused

by a large number of simultaneous switchings can cause considerable $L \cdot di/dt$ noise.

In addition to causing signal integrity problems, power supply noise results in variations in performance, which may also cause hold time or setup time violations. It is important to note that this effect varies from cycle to cycle, because power supply noise is strongly related to switching activities. Moreover, the supply voltage continues to scale with device scaling. The reduction in power supply voltage is accompanied by a decrease in noise margins as well. Consequently, modern designs are more prone to failures caused by power supply noise.

Increasing wire widths is a typical approach to counter *IR* drop. Besides wire sizing, decoupling capacitors are usually inserted to reduce *IR* drop and to suppress $L \cdot di/dt$ noise. Although on-chip decoupling capacitors are indispensable for robust power supply, it is important to note that they are usually realized as MOS capacitors. Hence, they are as leaky as transistors. For energy conservation, it is crucial to restrict the use of on-chip decoupling capacitors.

13.2.4 **Power dissipation**

As the clock network switches in every cycle, its total power dissipation can be significant. The IBM Power4 processor, for example, consumes 70% power in clock networks and latches [Anderson 2001]. Because of the high switching activity, dynamic power is the most significant component of clock power:

$$P_{clk} = CV_{DD}^2 f_{clk} = CV_{DD}^2 / C_p$$

The capacitive load C includes the gate capacitance of the sequential elements controlled by the clock signal, the interconnect capacitance of the clock network, and the capacitances associated with the buffers used in the clock network.

One very effective technique in reducing the switching capacitance of a clock network is through *clock gating* [Rabaey 2003]. One of many implementations of clock gating is to "AND" the clock signal with an enable signal, as shown in Figure 13.8. When the enable signal "EN" is low, all sequential elements in the

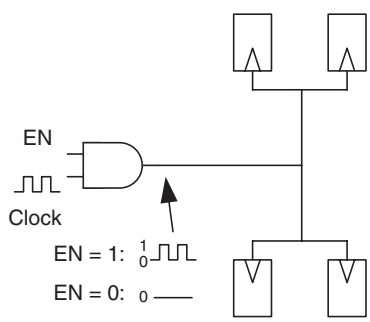


FIGURE 13.8

Clock gating for power reduction.

downstream of the AND gate will stay inactive. As the output terminals of these sequential elements remain quiet, the combinational logic between these sequential elements also stay inactive. In other words, the corresponding functional units have been disabled or clock gated. Because these sequential elements (and the ensuing combinational logic) do not participate in the switching activity, the switching capacitance of the clock network is reduced, effectively cutting down on the dynamic power dissipated by the clock network.

For clock gating to be effective, the sequential elements should be clustered such that a group of sequential elements can be gated by a single enable signal. It is not economical otherwise to have one AND gate for each sequential element. Moreover, there is an overhead of a control logic for the generation and distribution of enable signals for different clusters of clock-gated sequential elements. Propagating the clock signal through a mixture of buffers and AND gates further complicates the control of clock skew. Even for flip-flops that are not clock-gated, it may be necessary to insert an identical number of AND gates along the clock path to promote symmetry in the clock network.

Clock gating is also an active power management technique that further heightens the imbalance of switching activity at different locations of a chip. Time-varying variations such as temperature gradients may become more severe. The switching of a module between active and inactive modes also results in larger variations in the current flowing through the power supply network. All these variations are the main contributors to clock jitter.

13.2.5 Electromigration

Both the power supply network and the clock distribution network carry high currents because of the high capacitive load that they have to charge or discharge. Wires that carry high currents typically suffer from the effects of electromigration, where metal atoms migrate as a result of electrical current flowing through the metal material. Void appears in the area from where metal molecules migrate, leading to narrower line widths and eventually an open circuit. On the other hand, electromigration can also cause metal buildup in the form of hillocks in a wire, which may lead to a short with an adjacent wire. Metal lines that have existing cracks or other imperfections are particularly prone to electromigration. Circuit failure can also occur in the form of timing violation, because the resistance and capacitance of a metal wire change because of electromigration.

A good metric to characterize the reliability of a chip is the ***mean time to failure*** (MTTF). A mathematical model for the MTTF of a chip caused by electromigration is given by Black's equation [Black 1969]:

$$MTTF = (AJ^{-2})e^{E_a/kT}$$

where A is an empirically determined scaling factor, J is the current density, E_a is the activation energy that is determined by the material and its diffusion mechanism, k is Boltzmann's constant, and T is the temperature.

The typical current density at which electromigration occurs in modern on-chip interconnects is 10^6 to 10^7 A/cm^2. Because the current in a wire varies with time, the current density J in Black's equation should be replaced by the effective current density J_{eff}. For wires in a power supply network, the current flow is usually unidirectional (from a V_{DD} pad to a transistor). Thus, the effective current density is the average direct current density. For clock interconnects, the effective current in a wire is essentially zero, because the current flow is bidirectional. Treating the current as alternating, the root-mean-square current density should be used instead.

It is clear from Black's equation that the occurrence of electromigration in a design is mainly determined by the current density J and the temperature T. It is, therefore, desirable to reduce the switching activity of the circuit. We can also address the issue of electromigration by the following techniques that target the reduction of current density J. Increasing the wire width is the most straightforward way to decrease current density and increase MTTF. Multiple vias can be, and are often, used to improve reliability. The geometry of a via array should facilitate an even distribution of the current flow through all the vias. Turning corner with an oblique bend instead of a right-angled-bend also eliminates current crowding.

13.3 CLOCK NETWORK DESIGN

The two main subsystems in a clock network are: clock generation and clock delivery. Clock generation is an important topic that has been covered quite extensively in many textbooks on circuit design. This book, being an electronic design automation text, will focus on the synthesis of clock delivery networks. First, we cover some of the common clock topologies used in VLSI circuits. Second, we present the Elmore delay model, which is extensively used in the EDA community for the analysis and synthesis of clock networks. Third, we describe several basic clock synthesis algorithms, dealing with both skew scheduling and clock routing. Finally, we focus on a few fundamental clock optimization techniques, namely, buffer insertion, clock gating, wire sizing, and link insertion. For ease of presentation, the descriptions of many algorithms in this chapter may deviate from the original ones in the literature.

13.3.1 Typical clock topologies

The most economical way of distributing a clock signal is that of a tree topology. Assuming a binary tree structure, a clock tree with h levels of internal nodes can reach 2^h flip-flops at the leaf nodes.

An ideal clock tree is the *H-tree* topology [Fisher 1982; Kung 1982] as shown in Figure 13.9, where the basic building block at each level of the distribution network is a regular H-structure. Another scheme that yields equal-length interconnections is the X-tree (see also Figure 13.9), where the basic building

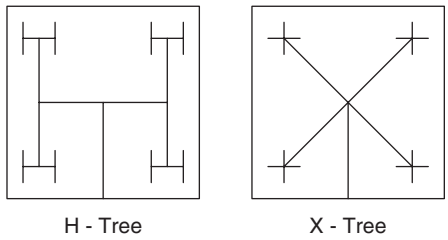


FIGURE 13.9

Symmetric H-tree and X-tree.

block is an X-structure [Bakoglu 1990]. All four corners of the H-structure and X-structure are equidistant from the center of the structure. Both the H-tree and X-tree achieve equal path length from the clock source to the leaf nodes by, respectively, repeating the H-structure and X-structure recursively top-down as shown in Figure 13.9. Such regular topologies also facilitate the addition of clock buffers in a symmetrical fashion.

H-trees (or X-trees), although effective in equalizing path lengths from a driver to a set of sinks, have serious limitations. These trees are best suited for regular layouts where the clock load is uniformly distributed over the entire chip. It is not particularly suitable for irregular placements with varying sink capacitances, which are common for cell-based designs. Moreover, a tree topology is more susceptible to the effects of variations in process parameters and operating condition because of its lack of redundancy; there exists only one unique path from the clock source to a flip-flop.

A very effective way of introducing redundant clock paths to a balanced clock tree is to add a trunk to connect all the leaf nodes of the clock tree, and branch off from the trunk to drive clock pins. In Figure 13.10, a balanced binary tree drives the center trunk at 4 positions uniformly distributed along the trunk. The binary tree is balanced, because the clock paths from the clock source to the 4 connecting points along the trunk are of equal length. The clock paths from the center trunk to the 5 clock pins are also designed to be of the same path length. The center trunk is of wider width, so that the associated resistivity is minimized. Consequently, the delays of the clock signal at various locations of the clock trunk can be minimized or considered to be uniformly equal. Hence, the clock trunk can provide a uniform clock reference for a region of the circuit.

The clock spines used in the first Intel Pentium 4 microprocessor [Kurd 2001] are variants of the clock trunk topologies. Figure 13.11 shows three clock spines that are driven at various points along the spines. The network of distributed clock buffers is almost a balanced binary tree, with the non-tree structure driven by each of the third level clock buffers being the exception. As in the case of a clock trunk topology, the spines provide a uniform clock

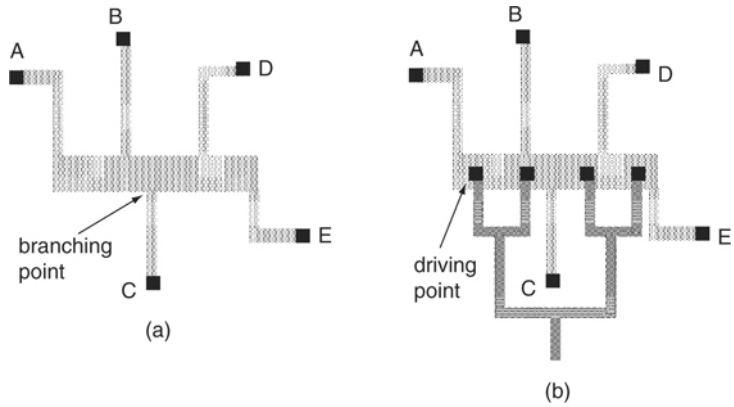


FIGURE 13.10

(a) A center trunk connecting 5 sinks. (b) A balanced binary tree driving the center trunk at 4 positions.

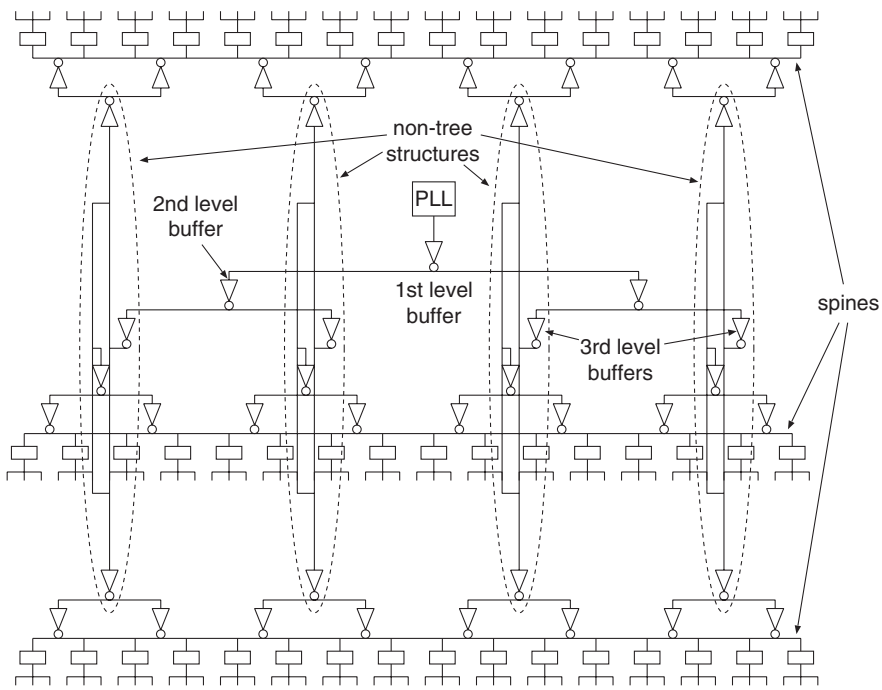


FIGURE 13.11

Clock spines in the first Intel Pentium 4 processor [Kurd 2001].

reference for all the clock trees radiating out of the spines. It is interesting to note that the clock signal goes through the same number of clock buffers before reaching the spines. Hence, the effects of device variations are minimized. However, clock skew or jitter at the clock buffers directly above the spines may result in short-circuit current between these buffers. In Figures 13.10 and 13.11, we show a trunk and a spine as a straight wire. This is an abstraction; in reality, the trunk or the spine could be a wire with multiple bends.

Compared with clock trunk topologies, a clock mesh provides significantly more alternative paths from the clock source to the flip-flops. A clock mesh is typically driven by distributed clock buffers. As shown in Figure 13.12, the clock buffers could either be located at the perimeter of the clock mesh, as in the case of the Alpha 21264 processor [Bailey 1998], or be uniformly distributed across the grid points of the clock mesh, as in the case of the IBM Power4 processor [Anderson 2001]. The distributed buffers for the mesh are typically driven by another clock network. For a mesh whose clock buffers are on the perimeter of the clock mesh, it is convenient to think of a boundary edge of the mesh as a clock trunk that connects these buffers. For a mesh whose clock buffers are uniformly distributed, it is typical to use an H-tree to distribute the clock buffers.

If a clock mesh that spans the entire chip has both high grid density and high clock buffer density, the clock signal can be considered to be available almost everywhere on the chip with minimal skew. Because of its high grid density,

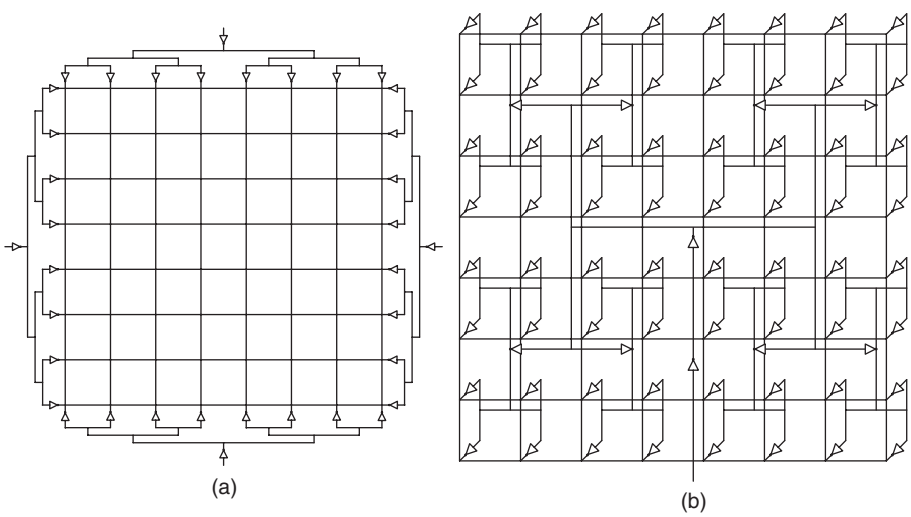


FIGURE 13.12

Two general clock mesh structures: (a) The clock drivers are at the perimeter. (b) The clock drivers are on the grid points.

most clock pins can be connected directly to the grid edges with short links. However, such a clock network incurs high power consumption. Furthermore, it is not amenable to clock gating, because most clock pins are connected to the clock mesh directly. A good compromise between power, skew, and wiring cost is to use a clock mesh with a moderate grid density and a moderate clock buffer density.

For such a clock mesh, the clock signal is no longer that freely available to flop-flops distributed across the chip. Additional clock subnetworks have to be constructed to connect them to the mesh. Figure 13.13 shows such a two-level clock distribution network: global (level) clock distribution and local (level) clock distribution. The global clock network distributes the clock signal from the clock source to various regions across the chip with a balanced H-tree driving a clock mesh. The local clock distribution subnetworks further distribute the clock signal to the flip-flops. Such a two-level design facilitates clock gating, especially when flip-flops that should be gated by the same control signal reside in close proximity.

Although this example shows a two-level clock network, it is not uncommon for a high-performance processor design to use a clock network with three or more levels of hierarchy. Different topologies can be deployed at each level. Instead of the use of a clock mesh as a global clock network, clock trunks and clock spines could also be used. Similarly, the local clock distribution subnetworks could either be a tree topology, clock trunk, clock spine, or even a clock mesh. Among these, trees are the topology of choice for the connection of the local flip-flops because of their low wiring cost. Because of the uneven

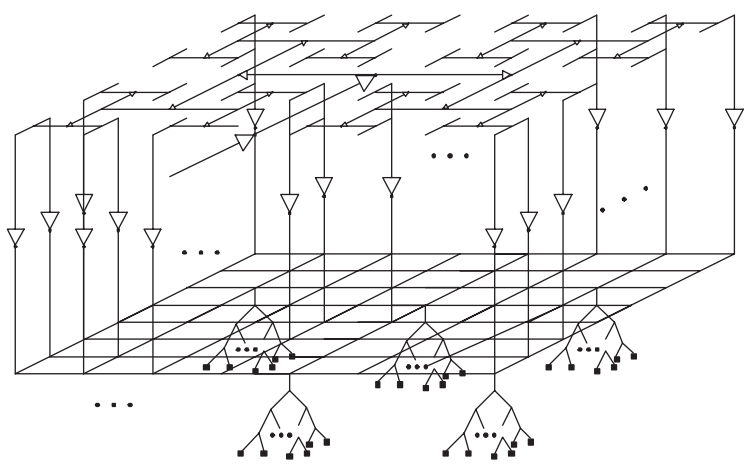
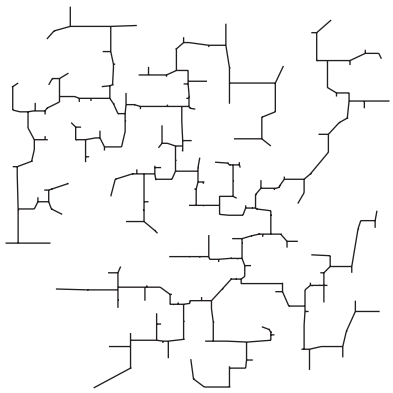


FIGURE 13.13

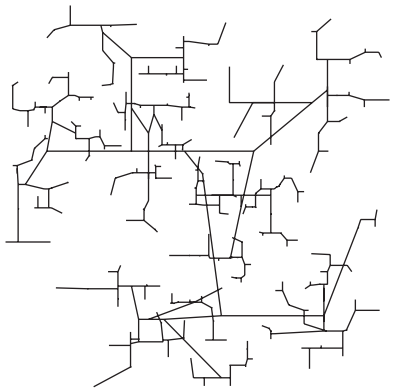
A hierarchical clock network with an H-tree driving a clock grid for global clock distribution, which in turn drives flip-flops through many local clock distribution networks.

distribution of flip-flops or clock load in a local region, it is common to use an asymmetric tree structure as the clock network. Figure 13.14 shows some asymmetric tree topologies constructed for the same set of clock pins under different skew requirements. As we can observe from the topologies, they become more complex and more costly as the skew constraints become more stringent (from (a) to (c)). The topologies shown here are an "abstraction" of the physical clock trees embedded in a Manhattan routing plane. Although they show the physical locations of internal nodes in the clock trees, the connections are not



Cost = 780

(a)

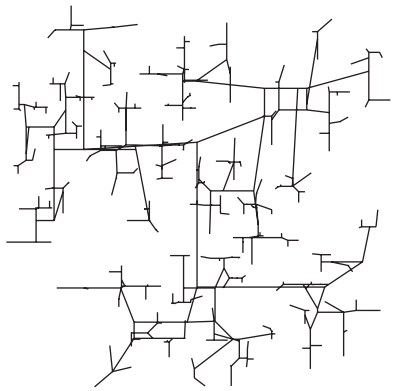


Cost = 1113

(b)

FIGURE 13.14

Topologies for the same set of clock pins under various kinds of skew constraints: (a) Very relaxed skew constraints. (b) Moderate skew constraints. (c) Very tight skew constraints.



Cost = 1230

(c)

FIGURE 13.14

(Continued)

embedded with horizontal and vertical wire segments. In other words, the connections are not shown with rectilinear routing.

Most studies on clock layout are concerned with the construction of a tree topology that satisfies the skew constraints. In fact, the synthesis of such a tree will be the main focus of Section 13.3.3. Non-tree structures, such as meshes or a hybrid of meshes and trees, typically can tolerate higher degrees of parameter variations in devices and wires, as well as uncertainties in system operating conditions because of its highly interconnected nature. In other words, the behavior of non-tree structures is more predictable. However, the robustness of a non-tree structure is achieved at the expense of higher routing cost compared with a tree topology. Consequently, tree structures are preferred over non-tree structures when routing area is a premium, but non-tree structures are the preferred solutions when the predictability of clock networks is critical.

Because deep-submicron designs are more susceptible to parameter variations, more and more designs have turned to non-tree structures for clock distribution. A good tradeoff between area and process variation is the insertion of cross-links in a tree topology, as shown in Figure 13.15. Here, wires are inserted into a tree to provide alternative paths between selected pairs of nodes whose skews are deemed to be highly susceptible to process variations. We will further elaborate on the synthesis of such a non-tree structure in Section 13.3.4.

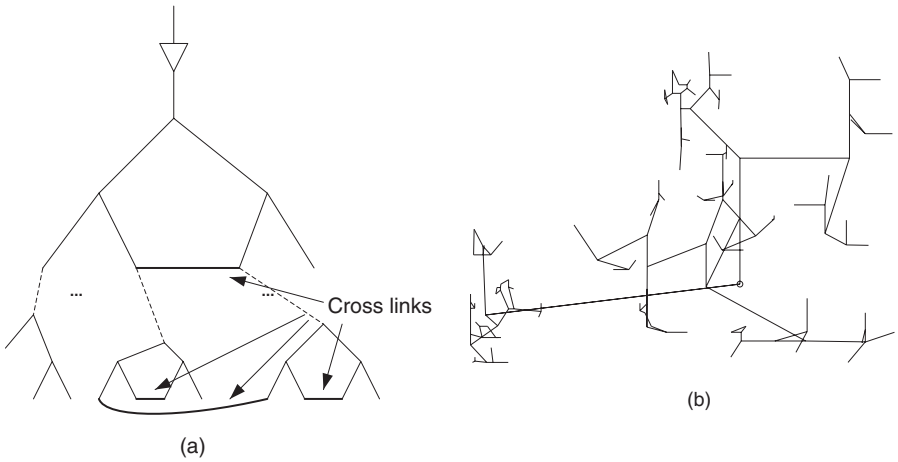


FIGURE 13.15

(a) An abstract clock tree with links inserted. (b) A clock layout for a benchmark circuit with cross-links inserted with the algorithm from [Lam 2005].

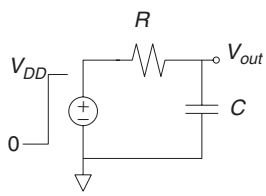


FIGURE 13.16

A simple RC circuit driven by a step input.

13.3.2 Clock network modeling and analysis

A key component to the synthesis of a reliable clock network is the modeling and analysis of clock signal delays. In this section, we describe a commonly used delay model for clock network synthesis, namely, the Elmore delay model [Elmore 1948]. In particular, we focus on the computation of Elmore delay for a tree.

Figure 13.16 shows a lumped RC circuit driven by a step input. The response at the capacitor is governed by the following expression:

$$v_{out}(t) = V_{DD}(1 - e^{-t/RC})$$

The time at which $v_{out}(t)$ reaches some specified critical voltage V_{crit} is given by

$$t_{crit} = RC \cdot \ln \frac{V_{DD}}{V_{DD} - V_{crit}}$$

In particular, the time at which the output voltage reaches 50% of V_{DD} is approximately $0.69 \times RC$. The rise time, which is typically defined to be the elapsed time between 10% and 90% of V_{DD}, is approximately $2.2 \times RC$.

It is, however, more difficult to calculate the 50% delay of a distributed line, or for that matter, an RC tree made up of lumped or distributed RC lines. Fortunately, a number of useful delay models have been developed to help approximate the delay of RC trees. Among these, the Elmore delay model is most commonly used.

First, we consider trees of lumped RC elements only. In [Elmore 1948], Elmore defined the delay of node i in an RC tree as

$$t_{Elmore,i} = \int_0^\infty t \cdot h_i(t)dt$$

where $h_i(t)$ is the *impulse response* to the unit impulse (applied at time 0) at time t, or equivalently, the derivative of the unit step response at time t. The 50% delay, denoted t_{50}, is the time for the monotonic step response to reach 50% of V_{DD}, and it is the median of the impulse response. In essence, the Elmore delay model uses the mean of the impulse response $h(t)$ to approximate the 50% delay of the step response.

Rubinstein, Penfield, and Horowitz [1983] derived an algorithmic approach to compute the Elmore delay of all nodes in an RC tree; the computation time is proportional to the number of lumped RC elements. Before giving the details of the approach, we first provide some definitions. Given an RC tree, there is a corresponding abstract topology. Figure 13.17 shows one such example. The tree is driven at its root (u) by a driver (or buffer), which is modeled as voltage source, labeled *src*, connected in series with an output resistance, R_u. Every internal node of the tree is connected to one or more child nodes. Because each of these connections (or tree edges) can be uniquely identified by the child node, we label the resistors between them according to the labels of the child nodes. Node u, for example, is connected to child nodes v and x by resistors R_v and R_x, respectively, in the RC tree. Each node i in the RC tree has an associated capacitor C_i.

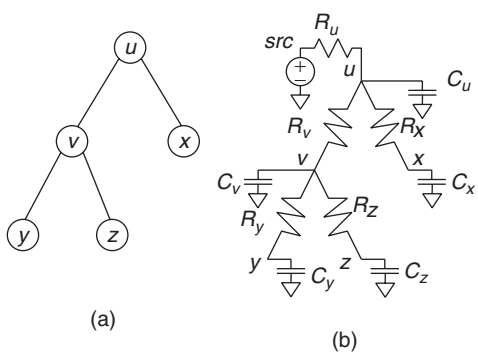


FIGURE 13.17

(a) An abstract topology. (b) The corresponding RC tree.

Let $Path(src, j)$ denote the path in the RC tree T from voltage source, src, to node j. Moreover, let $R_{Path(src,i) \cap Path(src,j)}$ denote the total resistance along the path common to $Path(src, i)$ and $Path(src, j)$. Furthermore, let T_i denote the subtree whose root node is i. The Elmore delay from src to i is given by the following two equivalent expressions:

$$\begin{aligned} t_{Elmore,i} &= \sum_{j \in T} C_j \times R_{path(src,i) \cap path(src,j)} \\ &= \sum_{R_k \in path(src,i)} R_k \sum_{j \in T_k} C_j \end{aligned} \tag{13.7}$$

The second expression, *i.e.,* Equation (13.7), gives a linear time computation of the Elmore delay. Consider the example in Figure 13.17, the Elmore delays of nodes v and z can be written as:

$$\begin{aligned} t_{Elmore,v} &= R_u(C_u + C_v + C_x + C_y + C_z) + R_v(C_v + C_y + C_z) \\ t_{Elmore,z} &= R_u(C_u + C_v + C_x + C_y + C_z) + R_v(C_v + C_y + C_z) + R_z C_z \end{aligned}$$

First, we observe that the capacitive terms in the parentheses are cumulative from right to left in the two preceding expressions or from bottom to top topologically if we traverse the RC tree. This suggests that in a bottom-up manner (or in a post-order traversal), we can compute at each node k the total downstream capacitance $C_{T_k} = \Sigma_{j \in T_k} C_j$. The procedure for the computation of downstream capacitance of node k in a bottom-up fashion is given in Algorithm 13.1.

Algorithm 13.1 Compute-C_T

Input: Node k

Output: Total downstream capacitance CT_k at node k

1. $C_{T_k} \leftarrow C_k$;
2. **for** each node $j \in$ Children(k) **do**
3. $\quad C_{T_k} \leftarrow C_{T_k} +$ Compute-$C_T(j)$;
4. **end for**
5. **return** C_{T_k};

Second, $t_{Elmore,z} = t_{Elmore,v} + R_z C_z$. In other words, the Elmore delay of a node is an accumulation of RC product terms from the voltage source to the node of interest. The RC product term at node k is that of the branch resistance R_k and the total downstream capacitance C_{T_k}. Therefore, in a top-down manner from the voltage source to node i, we can sum up the RC delay of each resistor along the path. In fact, in a top-down traversal of the tree T, we can compute the Elmore delays for all nodes. A description of the recursive computation of Elmore delays of nodes in T_k in a top-down fashion is given in Algorithm 13.2.

Algorithm 13.2 Compute-Elmore-Delay

Input: Node k

Output: Elmore delay of node k

1. **if** node k is the root node
2. $\quad t_{Elmore,k} \leftarrow 0$;
3. **else**
4. $\quad t_{Elmore,k} \leftarrow t_{Elmore,Parent(k)} + R_k \cdot C_{T_k}$;
5. **for** each $j \in Children(k)$ **do**
6. $\quad$ Compute-Elmore-Delay(j);
7. **end for**

On-chip interconnects are not lumped RC elements; they are distributed RC lines. Given an RC line whose resistance R and capacitance C are uniformly distributed over the line, we can divide the line evenly into n segments and model each segment as a lumped RC element, with resistance R/n and capacitance C/n. Assuming a step input at one end of the wire, the Elmore delay at the downstream endpoint is

$$\begin{aligned} t_{Elmore} &= R/n \cdot C + R/n \cdot C(n-1)/n + \cdots + R/n \cdot C/n \\ &= RC(n+1)/2n \end{aligned}$$

As n approaches ∞, t_{Elmore} approaches $RC/2$. We can replace a uniformly distributed RC line as a π-type lumped RC circuit as shown in Figure 13.18, where one-half of the wire capacitance is at the downstream end. Because the uniformly distributed RC line has to present a total capacitive load of C to all upstream resistors in the tree topology, the remaining half of the wire capacitance is at the upstream end of the wire.

If one end of the wire is connected to the clock pin of a flip-flop, the capacitance associated with that end should include both the gate capacitance of the clock pin and one-half of the wire capacitance.

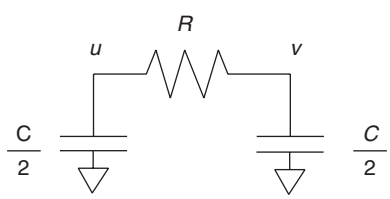


FIGURE 13.18

Modeling a uniformly distributed RC line of resistance R and capacitance C with a π-type circuit.

Remark: Is Elmore Delay Accurate Enough? The Elmore delay model has been used extensively to approximate the 50% delay point. It can easily be shown that the Elmore delay gives the 63% ($= 1 - 1/e$) delay of a simple RC circuit (with a single resistor and a single capacitor), which is an upper bound of the 50% delay. In general, the Elmore delay of a sink in an RC tree gives an upper bound on the actual 50% delay of the sink under not only the step input but also any monotonically increasing, piecewise-smooth input $u(t)$, with its derivative $\frac{d}{dt}u(t)$ being unimodal and symmetric [Gupta 1997]. The approximation of the 50% signal delay by the Elmore delay is exact only for a symmetric impulse response, where the mean is equal to the median [Gupta 1997]. Although the Elmore delay model may not be accurate, it has a high degree of *fidelity*. An optimal or near-optimal solution according to the estimator is also nearly optimal according to actual (SPICE-computed [Nagel 1975]) delay for routing constructions [Boese 1995] and wire sizing optimization [Cong 1996]. Studies in [Cong 1995b, 1998] also showed that the clock skew under the Elmore delay model has a high correlation with the actual (SPICE) skew. Figure 13.19 demonstrates the accuracy and fidelity of Elmore delay skew to actual skew for routing trees constructed under the Elmore delay [Cong 1998].

Of course, the Elmore delay model has a few disadvantages. First, the Elmore delay may not be very accurate. So, it is not suitable to be used directly for accurate circuit timing analysis. Also, it cannot handle the inductive effect, because the Elmore delay is defined for a monotonic response. More accurate delay estimation can be obtained with higher order moments.

13.3.3 Clock tree synthesis

Two problems relate to the synthesis of a clock net: (1) the determination of a *feasible clock schedule* that defines the arrival times of the clock signals at the

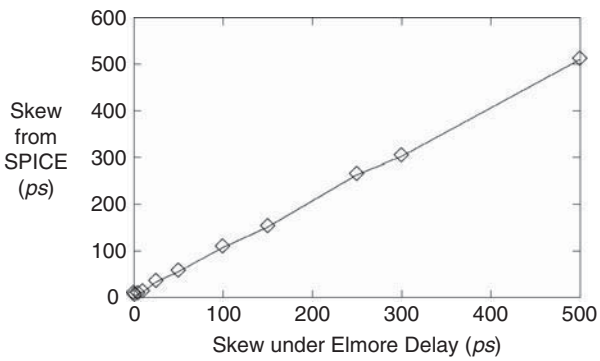


FIGURE 13.19

Elmore delay skew versus actual (SPICE simulation) delay skew for clock trees obtained by Greedy-BST/DME algorithm [Cong 1995b].

clock pins, and (2) the physical layout of the clock network that *realizes* the clock schedule. In the context of clock synthesis, a clock schedule is feasible if it meets the performance requirement without causing race hazards in the system operation. A physical clock network realizes the clock schedule if the clock signal arrives at the registers at the respective arrival times specified by the clock schedule. We refer to the first problem as that of *clock skew scheduling* and the second as that of *clock routing*.

13.3.3.1 *Clock skew scheduling*

Designers may impose additional constraints to make circuits more robust to process variations or have less power consumption. For example, we can make a circuit more robust to process variations by subtracting a *safety margin* from the upper bound constraint in Equation (13.5), or adding a safety margin to the lower bound constraint in Equation (13.6):

$$skew_{i,j} \leq C_p - t_{pFF}^{\max} - t_{logic}^{\max} - t_{setup}^{\max} - (T_i^{jitter} + T_j^{jitter}) - \delta_u \tag{13.8}$$

$$skew_{i,j} \geq t_{bold}^{\max} - t_{pFF}^{\min} - t_{logic}^{\min} + (T_i^{jitter} + T_j^{jitter}) + \delta_l \tag{13.9}$$

where $\delta_u \geq 0$ and $\delta_l \geq 0$ are the safety margins. In general, safety margins may vary for different pairs of flip-flops.

For convenience, we use $l_{i,j} \leq skew_{i,j} = t_i - t_j \leq u_{i,j}$ to represent lower- and upper-bound skew constraints between s_i and s_j and $\mathcal{C} = \{l_{i,j} \leq skew_{i,j} \leq u_{i,j}\}$ to denote the set of skew constraints for all sequentially adjacent clock pins s_i and $s_j \in S$, the set of all clock pins of flip-flops. A skew schedule X is an assignment of delay values t_i to each clock pin s_i. We say that X is feasible if $skew_{i,j}$ satisfies the skew constraints in $\mathcal{C}$.

The skew bounds, each constraining the difference of a pair of variables, can be represented by a constraint graph $G_{\mathcal{C}} = (V, E)$ as follows [Cormen 2001]: Each clock pin in S corresponds to a vertex in the constraint graph. For the two skew constraints associated with s_i and s_j, we generate two directed edges in $G_{\mathcal{C}}$, $e_{i,j}$ and $e_{j,i}$. The former edge captures the lower-bound constraint and the latter edge the upper-bound constraint. The weight of $e_{i,j}$, denoted by $w_{i,j}$, is $-l_{i,j}$, and the weight of $e_{j,i}$, denoted by $w_{j,i}$, is $u_{i,j}$. We will now give the motivation for such a graph formulation.

Consider a circuit with four flip-flops with clock pins $\{s_1, s_2, s_3, s_4\}$ for example. In this circuit, FF_1 feeds its output (through some combinational logic) to FF_2 and FF_3, both of which send data to FF_4. We illustrate in Figure 13.20 the corresponding constraint graph.

First, we consider the four upper-bound skew constraints among the four clock pins: $skew_{1,2} \leq u_{1,2}$, $skew_{2,4} \leq u_{2,4}$, $skew_{1,3} \leq u_{1,3}$, and $skew_{3,4} \leq u_{3,4}$. Now, the constraint $t_i - t_j \leq u_{i,j}$ can be written as $t_i \leq t_j + u_{i,j}$. With this expression, we can interpret t_i as a distance label of node s_i, and say that s_i is of a distance at most $u_{i,j}$ away from s_j. That is equivalent to the existence of a

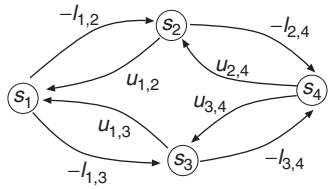


FIGURE 13.20

A constraint graph constructed from a circuit with four flip-flops represented by clock pins $\{s_1, s_2, s_3, s_4\}$. FF_1 is sequentially adjacent to FF_2 and FF_3, both of which are sequentially adjacent to FF_4.

directed edge $e_{j,i} \in E$ of weight $w_{j,i} = u_{i,j}$ from s_j to s_i for each constraint $t_i - t_j \leq u_{i,j}$, as shown in Figure 13.20.

Although FF_1 and FF_4 are not sequentially adjacent, the two data paths $FF_1 \rightarrow FF_2 \rightarrow FF_4$ and $FF_1 \rightarrow FF_3 \rightarrow FF_4$ induce two transitive constraints between s_1 and s_4:

$$\begin{aligned} skew_{1,2} + skew_{2,4} &= (t_1 - t_2) + (t_2 - t_4) = t_1 - t_4 \leq u_{1,2} + u_{2,4} \\ skew_{1,3} + skew_{3,4} &= (t_1 - t_3) + (t_3 - t_4) = t_1 - t_4 \leq u_{1,3} + u_{3,4} \end{aligned}$$

The tighter of the two constraints imposes an upper-bound constraint on the skew between s_1 and s_4:

$$skew_{1,4} = t_1 - t_4 \leq \min\{u_{1,2} + u_{2,4}, u_{1,3} + u_{3,4}\}$$

Consequently, the upper bound on $skew_{1,4} = t_1 - t_4$ is essentially the shortest path distance from s_4 to s_1 in the constraint graph.

Next, we consider the lower bound constraints $l_{1,2} \leq skew_{1,2}$, $l_{2,4} \leq skew_{2,4}$, $l_{1,3} \leq skew_{1,3}$, and $l_{3,4} \leq skew_{3,4}$. Because $l_{i,j} \leq t_i - t_j = skew_{i,j}$ is equivalent to $t_j - t_i \leq -l_{i,j}$, we add a directed edge $e_{i,j} \in E$ of $w_{i,j} = -l_{i,j}$ from s_i to s_j, as shown in Figure 13.20. It is obvious that $skew_{1,4}$ is bounded from below by the negative of the shortest distance from s_1 to s_4.

It can be shown that there exists a feasible skew schedule subject to $\mathcal{C}$ if, and only if, G_C has no negative-weight cycles [Cormen 2001]. To perform a feasibility check and to compute a feasible skew schedule, we first augment the constraint graph $G_C = (V, E)$ with a source vertex S_0 and connect S_0 to each $s_i \in V$ with an outgoing edge of zero weight. Then, we apply the Bellman-Ford algorithm [Cormen 2001] to the augmented graph to compute the shortest distances from S_0 to all vertices. If the constraint graph has no negative-weight cycles, the Bellman-Ford algorithm terminates with appropriate shortest distances (or t_i's) assigned to clock pins; it returns a flag indicating that the schedule is infeasible otherwise. A careful implementation of the Bellman-Ford algorithm for the feasibility check has $O(|V||E|)$ time complexity.

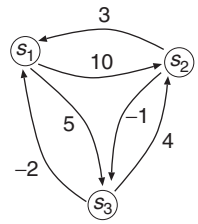
The preceding formulation checks for the feasibility of a given clock frequency (or clock period C_P). We can perform clock frequency optimization (or minimization of clock period) by building on the preceding formulation with an iterative binary search procedure. Suppose we are to determine the smallest feasible C_P within the range $C_{P,\min} \leq C_P \leq C_{P,\max}$. First, we consider $C_P = (C_{P,\min} + C_{P,\max})/2$ and formulate the set of skew constraints C accordingly. If the Bellman-Ford algorithm returns a feasible schedule, we update $C_{P,\max}$ to $(C_{P,\min} + C_{P,\max})/2$; otherwise, we update $C_{P,\min}$ to $(C_{P,\min} + C_{P,\max})/2$. We repeat the binary search procedure until $C_{P,\min}$ and $C_{P,\max}$ are sufficiently close.

Another important concept related to skew scheduling in the context of clock tree synthesis is that of a *feasible skew range*. Consider a circuit with three clock pins $\{s_1, s_2, s_3\}$ of three flip-flops FF_1, FF_2, and FF_3 forming a cyclic data path $FF_1 \rightarrow FF_2 \rightarrow FF_3 \rightarrow FF_1$. The skew constraints among the three flip-flops $-10 \leq skew_{1,2} \leq 3$, $-5 \leq skew_{1,3} \leq -2$, and $1 \leq skew_{2,3} \leq 4$ are captured in the constraint graph as shown in Figure 13.21a. We refer to the range $[l_{i,j}, u_{i,j}]$ defined by the lower and upper bounds of $skew_{i,j}$ as the skew range of $skew_{i,j}$.

Interestingly, even though zero-skew is within the skew ranges of $skew_{1,2}$ and $skew_{2,3}$, a feasible skew schedule does not exist if we choose $skew_{1,2} = 0$. This is because of the transitive skew constraints $-9 \leq t_1 - t_2 \leq -3$ between s_1 and s_2 induced by the skew constraints of $skew_{1,3}$ and $skew_{2,3}$. Such stricter constraints are, in fact, captured by the shortest paths between s_1 and s_2 in the constraint graph: the upper bound of $skew_{1,2}$ is essentially the shortest path distance from s_2 to s_1 in the constraint graph, and $skew_{1,2}$ is bounded from below by the negative of the shortest distance from s_1 to s_2.

Consider a constraint graph with no negative-weight cycles. Let $d_{i,j}$ denote the shortest distance to s_j from s_i. The inequalities $-d_{i,j} \leq skew_{i,j} \leq d_{j,i}$ define the feasible skew range of $skew_{i,j}$, denoted as $FSR_{i,j}$. Given a constraint graph G_C with no negative cycles, we can build an all-pairs shortest distance matrix $D = \{d_{i,j} : s_i, s_j \in S\}$ from G_C in $O(|V|^3)$ by the Floyd-Warshall algorithm [Cormen 2001] to represent the feasible skew ranges of all skew pairs. Figure 13.21b shows the all-pairs shortest distance matrix of the original constraint graph in Figure 13.21a.

We say that a *skew commitment* is made when we narrow a nontrivial *FSR* to a single skew value. In fact, we can commit $skew_{i,j}$, the skew between s_i and s_j, to any $x \in [-d_{i,j}, d_{j,i}] = FSR_{i,j}$, because a feasible skew schedule that contains such a skew commitment always exists. However, we may not make two or more such skew commitments independently without affecting the existence of a feasible skew schedule. Algorithm 13.3 gives an $O(|V|^2)$ approach for updating matrix D after we commit $skew_{i,j}$ to a skew value of x by shrinking the feasible skew range to a single value. In general, the algorithm can be applied whenever we refine the skew range of a pair of clock pins. Figure 13.21c shows the resultant matrix D after a skew commitment of $skew_{1,2} = -3$.



Constraint graph G_C

(a)

	s_1	s_2	s_3
s_1	0	9	5
s_2	−3	0	−1
s_3	−2	4	0

All-pairs shortest distance matrix D

(b)

	s_1	s_2	s_3
s_1	0	3	2
s_2	−3	0	−1
s_3	−2	1	0

Updated matrix D after committing $skew_{1,2} = -3$

(c)

FIGURE 13.21

An example showing the incremental skew scheduling based on an all-pairs shortest-distance matrix: (a) The constraint graph G_C. (b) The all-pairs shortest distance matrix D. (c) The all-pairs shortest distance matrix after committing the skew of s_1 and s_2 to $skew_{1,2} = -3$.

We now construct an $O(|V|^3)$ algorithm for incremental scheduling. First, the Floyd-Warshall algorithm is used to generate matrix D in $O(|V|^3)$ time complexity. As long as there exist uncommitted skews (or nontrivial feasible skew ranges), we select one of them and commit the skew to an arbitrary value in the feasible skew range. Then, we apply the incremental update procedure in

Algorithm 13.3 Update All-Pairs Shortest Distance Matrix

Input: A skew commitment $skew_{i,j} = x$, an all-pairs shortest distance matrix $D = \{d_{i,l}\}$

Output: An updated matrix D

1. Set $d_{i,j} = -x$ and $d_{j,i} = x$;
2. **for** each $d_{k,l}, 1 \leq k \neq l \leq n$ in D **do**
3. Set $d_{k,1} = \min\{d_{k,l}, d_{k,i} - x + d_{j,l}, d_{k,j} + x + d_{i,l}\}$;
4. **end for**

Algorithm 13.3. In the worst case, we have to perform $|V| - 1$ skew commitments, hence the $O(|V|^3)$ time complexity.

In a sense, the incremental skew scheduler finds a spanning tree of the complete graph represented by the matrix D. Here, a skew commitment is essentially an edge of the spanning tree. (In reality, there are two edges for each skew commitment, with the edge weight being the committed skew value or the negation of it, depending on the direction of the edge.) Consequently, after at most $n - 1$ skew commitments, the spanning tree connects all sinks in the graph. Moreover, the skew between any pair of clock sinks is defined by the sum of the edge weights along the path between the two corresponding vertices in the spanning tree.

Remark: Clock Scheduling and Power Supply Noise: Very recent works on clock schedule optimization also considered current-induced noise control and minimization. If not minimized, current induced noise, which includes *IR* drop and $L \cdot di/dt$ noise, has an adverse effect on circuit performance, especially for low-power systems with reduced supply voltage, because low noise margins are a primary concern. An effective method to reduce the current-induced noise is to design the circuit such that the current drawn is fairly stable without large peaks. In many designs with a huge clock distribution network, current peaks are usually caused by the simultaneous switching of highly loaded clock lines, as well as by the switching activities in the sequential elements when they are triggered and in the ensuing combinational logic. There are several skew scheduling algorithms for spreading out these switching activities over time [Vuillod 1996; Vittal 1996; Lam 2002, 2003; Yu 2007]. The reader may refer to the relevant papers for more details.

13.3.3.2 *Clock tree routing*

We now deal with the problem of clock tree routing (*i.e.,* the construction of a tree topology that realizes the specified skew schedule). Recall that S contains the set of clock pins of flip-flops $\{s_1, \ldots, s_n\}$. For convenience, we also include the clock source, denoted as s_0 in S. In general, the clock routing problem can be formulated as follows: Given $\{l_{s_1}, \ldots, l_{s_n}\}$, a set of locations of the clock pins $\{s_1, \ldots, s_n\}$ of flip-flops and skew constraints on various pairs of flip-flops,

construct with minimum wiring cost, a clock tree that satisfies the setup time and hold-time skew constraints in Equations (13.8) and (13.9). The location of the clock source, denoted as l_{s_0}, may also be given. If l_{s_0} is not given, the clock router will also determine a convenient location for s_0. There are possibly other constraints and/or objectives to the problem:

1. We want to impose a constraint on the rise/fall times of the clock signal at the sinks, because it is critical to keep the clock signal waveform clean and sharp.
2. We want the clock network to be tolerant of process variations, which cause the wire widths and device sizes on the fabricated chip to differ from the specified wire widths and device sizes, respectively, resulting in so-called *process skew* (*i.e.,* clock skew caused by process variations).

Several simplifications to the clock routing problem were made in the past to make the task easier. It is the enabling concept of (*absolute*) *global skew* that makes these simplifications possible. Indeed, these simplifications cultivate a prolific area of studies on clock tree routing.

The global skew is defined to be the maximum among all the absolute skew values between clock pins: $gskew = \max_{i,j}|t_i - t_j|$. Several simpler versions of the clock-routing problem centered around the concept of global skew are given in the following:

1. Given a set of sink locations, construct with minimum routing cost a clock routing that minimizes the global skew *gskew*.
2. Given a set of sink locations, construct with minimum routing cost a clock routing that achieves zero skew (*i.e.,* $gskew = 0$). In other words, all clock pins are required to have an identical clock delay. This is known as the zero-skew routing problem.
3. Given a set of sink locations and a skew bound $B \geq 0$, construct with minimum routing cost a clock routing that satisfies $gskew \leq B$. This is called the bounded-skew routing problem.

It is through these simplifications that we can obtain a better understanding of the general clock-routing problem, which is also known as the useful skew–routing problem, defined at the beginning of this section. We will now describe various algorithms on clock routing on the basis of the following classification: (1) zero-skew routing, (2) bounded skew routing, and (3) useful-skew routing.

In the descriptions of these algorithms, we distinguish a physical interconnect tree T from its *abstract topology* G, which is a binary tree such that all sinks are the leaf nodes of the binary tree. Every non-leaf node of G has two child nodes, with a possible exception that the root node may have only one child node. We first introduced the concept of an abstract topology in Section 13.3.2. We will now provide a more formal definition. The source driver, denoted s_0, is the root node of the tree. When l_{s_0}, the location of s_0, is given, s_0 has a singleton internal node, denoted as s_0', as its only child; otherwise, s_0

has two child nodes. Each non-root node v is connected to its parent, denoted as $p(v)$, by edge e_v. Consider any two nodes, say u and v, with a common parent node $w = p(u) = p(v)$ in the abstract topology. The signal from the source has to pass through w before reaching u and v (and their descendants).

The embedding of an abstract topology G to form an interconnect tree T involves the mapping of each internal node $v \in G$ to a location $l_v = (x_v, y_v)$ in the Manhattan plane, where (x_v, y_v) are the x- and y-coordinates, and replacing each edge $e \in G$ by a rectilinear edge or path. It is important to note that most clock-routing algorithms are concerned with only the mapping of internal nodes to physical locations and not the actual routing of wires, a task that could be accomplished by a maze router. Figure 13.22 shows an abstract topology and three of its many possible ways of embedding. Note that s_0' is the singleton child node of s_0 because the location of s_0 is given. The subtree rooted at node v in T is denoted as T_v. In T, the cost of edge e_v is its wire length, denoted by $|e_v|$. The cost of a routing tree T is the sum of its edge costs. Among the three ways of embedding the abstract topology in Figure 13.22, the embedding in (d) has the highest cost.

The objective of a clock routing algorithm is therefore twofold: to generate an abstract topology and to embed the abstract topology. For each of the clock-routing problems, namely zero-skew routing, bounded-skew routing, and useful-skew routing, we will first describe how we can embed a given abstract topology to satisfy the respective skew constraint(s). Then, we will present approaches to generate an abstract topology together with embedding.

13.3.3.3 *Zero-skew routing*

The problem of zero-skew routing was first successfully solved in [Tsay 1991] with a bottom-up approach. The *Deferred-Merge Embedding* (DME) algorithm, proposed independently in [Edahiro 1991; Chao 1992; Boese 1992], generalized the approach in [Tsay 1991] with the enabling concept of a *merging segment* to achieve zero-skew routing for a given abstract topology.

In general, given two zero-skew trees, there exists a set of locations at which two zero-skew trees can be joined with the minimum wire length such that the new tree is also of zero skew. In Figure 13.23b, for example, any point l_a on the line segment ms_a is equidistant from sinks s_1 and s_2 (*i.e.,* we obtain a zero-skew subtree rooted at l_a with sinks s_1 and s_2). For ease of illustration, the zero-skew tree in Figure 13.23 is constructed under the path length delay model. The model uses the wire length of the unique path between an ancestor node and a descendant node in the physical routing tree to estimate the delay between the two nodes. Many illustrations of different clock-routing algorithms in this section are based on the path length delay model.

Given S and an abstract topology G, the DME algorithm exploits this flexibility and embeds internal nodes of G by means of a two-phase approach: (1) a bottom-up phase that constructs for each internal node of G a set of possible placement locations of the node in a zero-skew tree T; and (2) a top-down

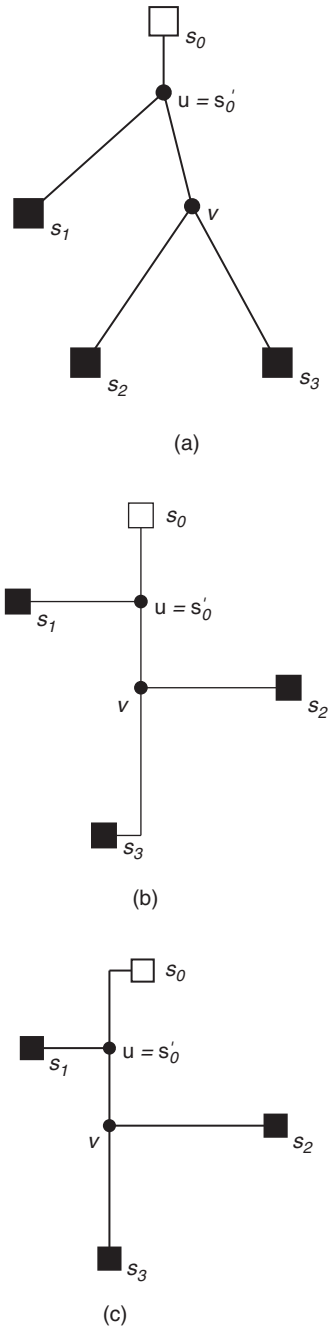
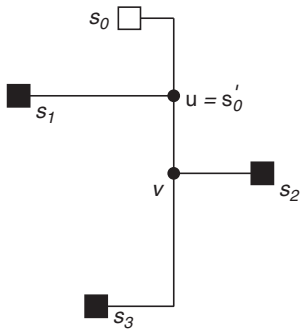


FIGURE 13.22

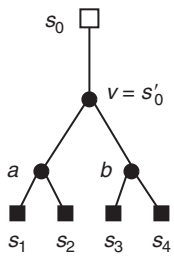
(a) An abstract topology. (b)–(d) Three different ways of embedding the topology.



(d)

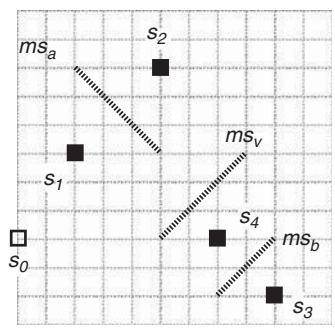
FIGURE 13.22

Continued



Topology

(a)



Bottom-up Merging phase

(b)

FIGURE 13.23

An illustration of the bottom-up phase of the DME algorithm: (a) Topology of a clock source s_0 and 4 sinks $s_{1..4}$. (b) Merging segments of internal nodes *a, b* and $v = s_0$.

embedding phase that determines for each internal node its exact location in T. Note that we do not perform actual Manhattan routing of the connections between nodes in the clock tree.

We refer to the set of possible placement locations of $v \in G$ in a minimum-cost zero-skew tree as the merging segment of v, denoted as ms_v. The segment ms_v is always a line segment (with possibly zero length when it degenerates into a point) with slope $+1$ or -1, as shown in Figure 13.23. This stems from the fact that a circle in the Manhattan routing plane is a square rotated by 45 degrees.

For each leaf node $s_i \in G$, the merging segment of s_i is simply l_{s_i}, the location of sink s_i. Suppose v is the parent node of s_i and s_j, which are at a distance of $d(l_{s_i}, l_{s_j})$ apart. Under the path length delay model, any point in ms_v should be of distance $d(l_{s_i}, l_{s_j})/2$ from l_{s_i} and l_{s_j}. The segment ms_v can be computed by taking the intersection of two Manhattan circles, both of radius $d(l_{s_i}, l_{s_j})/2$, that are centered at l_{s_i} and l_{s_j}. Because a Manhattan circle is a square that is rotated 45 degrees, the intersection must be a line segment of slope $+1$ or -1. Because it lies on the circumference of a Manhattan circle, we refer to such a line segment as a *Manhattan arc*.

Now, we consider a more general case. Let a and b be the child nodes of $v \in G$. The construction of ms_v depends on ms_a and ms_b, hence the bottom-up processing order. In particular, any point on ms_v should allow a and b to be merged with *minimum* wiring cost $|e_a| + |e_b|$, while maintaining zero skew among all sinks in T_v.

Let L denote the shortest Manhattan distance between ms_a and ms_b (*i.e.,* $d(ms_a, ms_b) = L$). Here, the distance between two sets of points P and Q is defined as $d(P, Q) = \min\{d(p, q) | p \in P, q \in Q\}$. Let ms_v be at a distance of $x \cdot L$ from ms_a where x is between 0 and 1. Given t_a, the Elmore delay from a to its sinks in T_a, the delay from v to sinks in T_a is

$$t_a + r \cdot x \cdot L \cdot \left(C_{T_a} + \frac{c \cdot x \cdot L}{2} \right)$$

where C_{T_a} is the total capacitance of the subtree T_a, and r and c are, respectively, the unit-length resistance and capacitance of a wire of some prespecified width. When T_a happens to be just a clock pin s_i, C_{T_a} is the gate capacitance associated with clock pin and $t_a = 0$. Similarly, the Elmore delay from v to sinks in T_b is

$$t_b + r \cdot (1 - x) \cdot L \cdot \left(C_{T_b} + \frac{c \cdot (1 - x) \cdot L}{2} \right)$$

where t_b is the Elmore delay from b to its sinks in T_b, and C_{T_b} is the total capacitance of the subtree T_b. We can now solve for x as follows [Tsay 1991]:

$$x = \frac{t_b - t_a + r \cdot L \cdot (C_{T_b} + c \cdot L/2)}{r \cdot L \cdot (c \cdot L + C_{T_a} + C_{T_b})}$$

If $0 \leq x \leq 1$, we have found $|e_a| = x \cdot L$ and $|e_b| = L - |e_a|$. Clearly, the wiring cost to merge ms_a and ms_b is L, which is the minimum possible. Consequently, $C_{T_v} = C_{T_a} + C_{T_b} + c \cdot (|e_a| + |e_b|)$. Note that the preceding computation assumes both edges e_a and e_b have the same width. A simple extension can be made to achieve zero-skew merging even when e_a and e_b have different widths.

If $x < 0$ or $x > 1$, it implies that the wire delay in a wire of length L is insufficient to balance the delay difference in t_a and t_b. It is, therefore, necessary to use a wiring cost $|e_a| + |e_b| > L$ to achieve zero-skew. Without loss of generality, let $t_a > t_b$. Then, $|e_a| = 0$, and $|e_b|$ is obtained by solving the following equation [Tsay 1991]:

$$t_a = t_b + r \cdot |e_b| \cdot \left(C_{T_b} + \frac{c \cdot |e_b|}{2} \right)$$

Given $|e_a|$ and ms_a, the union of all Manhattan circles of radius $|e_a|$ that are centered at points on ms_a gives us a tilted rectangular region (a rectangle rotated by 45 degrees). All points in the tilted rectangular region, referred to as trr_a, are at a distance of at most $|e_a|$ from ms_a. Given $|e_b|$ and ms_b, trr_b can be similarly obtained. The intersection of trr_a and trr_b is the merging segment for v, which is again a Manhattan arc, as shown in Figure 13.24. Figure 13.24a shows an example for the case when $|e_a| + |e_b| = d(ms_a, ms_b)$, whereas Figure 13.24b shows an example for the case when $|e_a| = 0$ and $|e_b| > d(ms_a, ms_b)$. In the latter example, ms_v is a subset of ms_a, and it resides completely within trr_b.

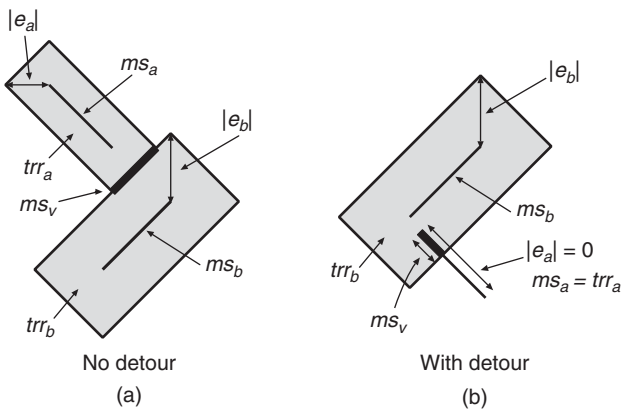


FIGURE 13.24

Intersection of trr_a and trr_b to obtain ms_v.

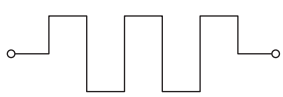


FIGURE 13.25

Snaking to lengthen the connection between two points.

Depending on the embedded locations of v and b, $|e_b|$ may be longer than $d(l_v, l_b)$. In that case, wire snaking or detour wiring, as shown in Figure 13.25, is necessary to implement the desirable wire length of $|e_b|$ in the final clock layout.

The bottom-up merging process computes a tree of merging segments. The process stops when we have computed ms_{s_0}, the merging segment of s_0 if l_{s_0} is not given; otherwise, the process terminates when we have computed the merging segment of s_0', the singleton child node of s_0. We refer to such a tree as a *merging tree*. Every node $v \neq s_0$ or s_0' in the merging tree also has an associated wiring cost $|e_v|$ for its connection to its parent node.

Given the merging tree, the root node s_0 is first embedded if the clock source location is not given. In such a case, any point on ms_{s_0} can be selected to be l_{s_0}; otherwise, we pick any point on $ms_{s_0'}$ that is closest to l_{s_0} to embed s_0'. Now, we proceed in a top-down fashion to embed other internal nodes of G. An internal node v is embedded at any point on ms_v that is of distance no farther than $|e_v|$ from $l_{p(v)}$, the embedded location of its parent node $p(v)$. Hence, v could be embedded at any point that lies on the intersection of the merging segment ms_v and the tilted rectangular region bounded by a Manhattan circle whose center is $l_{p(v)}$ and radius is $|e_v|$. The DME algorithm is summarized in Algorithm 13.4.

Algorithm 13.4 Deferred-Merge Embedding (DME)

Input: A set of clock pins (possibly including clock source s_0) S, an abstract topology G

Output: A zero-skew clock tree routing T

1. Initialize $ms_{s_i} = l_{s_i}$ for all $s_i \in S$;
2. **for** each merging of two subtrees T_a and T_b to form T_v based on bottom-up topological sort of abstract topology G **do**
3. Compute $|e_a|$ and $|e_b|$;
4. Construct trr_a from ms_a and $|e_a|$ and trr_b from ms_b and $|e_b|$;
5. $ms_v \leftarrow trr_a \cap trr_b$;
6. **end for**
7. **if** location of clock source s_0 is given
8. Choose $l_{s_0'} \in ms_{s_0'}$ such that $d(l_{s_0'}, l_{s_0}) = d(ms_{s_0'}, l_{s_0})$;
9. **else**
10. Choose any $l_{s_0} \in ms_{s_0}$;
11. **for** each remaining internal node $v \in G$ in top-down order **do**
12. Let $p(v)$ be the parent node of v;
13. Choose $l_v \in ms_v$ such that $d(l_v, l_{p(v)}) \leq |e_v|$;
14. **end for**

Figure 13.23 gives an example of a clock net with four sinks s_1–s_4. For the topology given in Figure 13.23a, the DME algorithm constructs merging segments ms_a, ms_b, and ms_v of the internal nodes a, b, and v, respectively, under the path length delay model as shown in Figure 13.23b. In this example, v is also s_0', the singleton child node of s_0, whose location is given. Each of the non-root internal nodes is embedded at a point on its merging segment that is closest to its parent node as shown in Figure 13.26.

Example 13.2 Zero-Skew Routing under Elmore Delay Model. Consider an example with 4 clock sinks [Tsay 1991], as shown in Figure 13.27b. Sink s_1 is at $(8, 0)$ with sink capacitance $c^g_{s_1} = 16$ F; sink s_2 is at $(22, 6)$ with $c^g_{s_2} = 10$ F; sink s_3 is at $(0, 10)$ with $c^g_{s_3} = 1$ F; sink

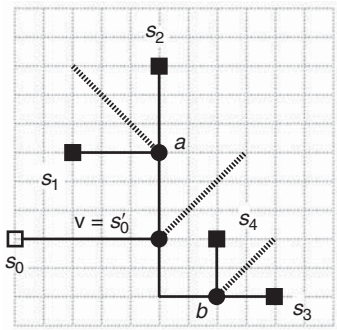


FIGURE 13.26

An illustration of the top-down phase of the DME algorithm: Zero-skew clock tree with a total wire length of 22 units for the example in Figure 13.23.

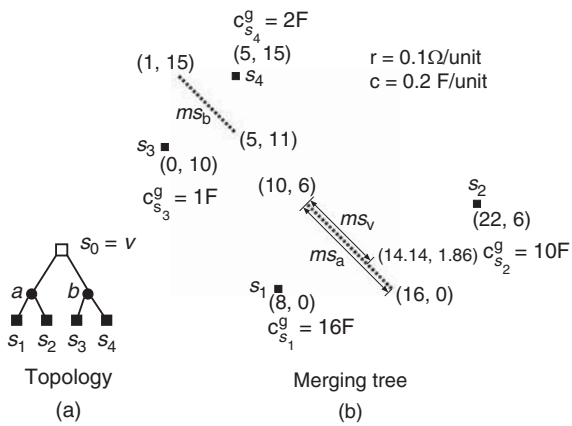


FIGURE 13.27

Application of DME on a 4-sink example under the Elmore delay model: (a) Abstract topology of 4 sinks. (b) Merging segments of internal nodes a, b, and $s_0 = v$.

s_4 is at $(5, 15)$ with $c^g_{s_4} = 2$ F. In this example, we assume unit-length wire resistance r and unit-length wire capacitance c to be, respectively, 0.1Ω and 0.2 F.

The abstract topology G under which a zero-skew routing tree for the 4 sinks is to be constructed is given in Figure 13.27a. Because the location of the clock source is not given, the topology is a strictly binary tree; the root $s_0 = v$ has two child nodes a and b, which, respectively, connect sink pair s_1 and s_2 and sink pair s_3 and s_4.

We now consider the merging of s_1 and s_2 (*i.e.,* the computation of ms_a). The distance between s_1 and s_2 is $L = d(l_{s_1}, l_{s_2}) = 20$ units. Let $x \cdot L$ be the distance of ms_a from s_1. Because the child nodes of a are sinks, the delays of the sink nodes are $t_1 = t_2 = 0$ s, and the total capacitances in T_{s_1} and T_{s_2} are, respectively, $C_{T_{s_1}} = c^g_{s_1} = 16$ F and $C_{T_{s_2}} = c^g_{s_2} = 10$ F. We can solve for x as follows:

$$\begin{aligned} x &= \frac{t_2 - t_1 + r \cdot L \cdot (C_{T_{s_2}} + c \cdot L/2)}{r \cdot L \cdot (c \cdot L + C_{T_{s_1}} + C_{T_{s_2}})} \\ &= \frac{10 + 2}{4 + 16 + 10} \\ &= 0.4 \end{aligned}$$

Therefore, $|e_{s_1}| = 0.4 \cdot 20 = 8$ units and $|e_{s_2}| = (1 - 0.4) \cdot 20 = 12$ units. We compute the merging segment ms_a by computing the intersection of the tilted rectangular regions trr_{s_1} and trr_{s_2}. The tilted rectangular region trr_{s_1} is defined by the coordinates $(0, 0)$, $(8, 8)$, $(16, 0)$, and $(8, -8)$. Essentially, it is a Manhattan circle with its center at $l_{s_1} = (8, 0)$ and a radius of 8 units. The tilted rectangular region trr_{s_2} is a Manhattan circle with its center at $l_{s_2} = (22, 6)$ and a radius of 12 units. The intersection of trr_{s_1} and trr_{s_2} gives us ms_a, which is a Manhattan arc defined by the coordinates $(10, 6)$ and $(16, 0)$, as shown in Figure 13.28.

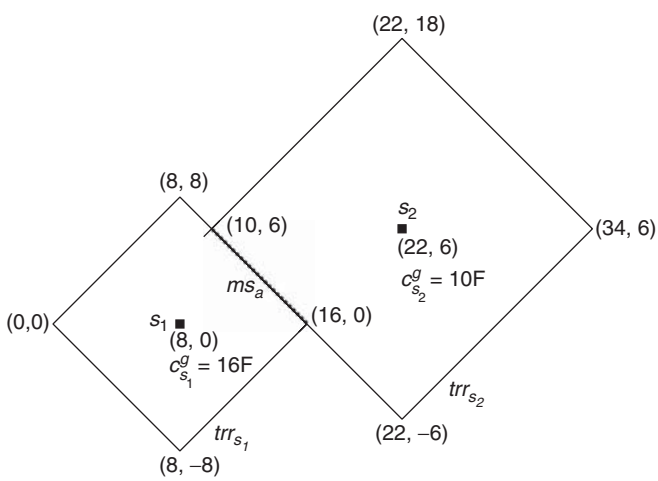


FIGURE 13.28

Computation of ms_a.

If we embed the internal node a at any point on ms_a and connect it to s_1 and s_2 with the shortest connections possible, the sink delays t_1 and t_2 from a under the Elmore delay model are both 13.44 ns, which can be computed as follows:

$$\begin{aligned} t_1 &= r \cdot x \cdot L \cdot \left(C_{T_{s_1}} + \frac{c \cdot x \cdot L}{2} \right) \\ &= 0.1 \cdot 8 \cdot (16 + 8 \cdot 0.2/2) \\ &= 13.44 \text{ ns}, \\ t_2 &= r \cdot (1-x) \cdot L \cdot \left(C_{T_{s_2}} + \frac{c \cdot (1-x) \cdot L}{2} \right) \\ &= 0.1 \cdot 12 \cdot (10 + 12 \cdot 0.2/2) \\ &= 13.44 \text{ ns} \end{aligned}$$

Similarly, we can compute the merging segment ms_b for the merging of s_3 and s_4. The merging segment ms_b, whose endpoints are $(1, 15)$ and $(5, 11)$, is shown in Figure 13.27b. With shortest connections to s_3 and s_4 from any point on ms_b, the sink delays t_3 and t_4 from b are both 0.96 ns.

Now, let us consider the merging of T_a and T_b. Besides r and c, the following parameters are required to determine the location of the merging segment ms_v: $t_a = t_1 = t_2 = 13.44$ ns, $t_b = t_3 = t_4 = 0.96$ ns, $L = d(ms_a, ms_b) = 10$ units, $C_{T_a} = c^g_{s_1} + c^g_{s_2} + 0.2 \cdot 20 = 16 + 10 + 4 = 30$ F, and $C_{T_b} = c^g_{s_3} + c^g_{s_4} + 0.2 \cdot 10 = 1 + 2 + 2 = 5$ F. Defining $x \cdot L$ to be the distance of ms_v from ms_a and solving for x, we obtain

$$\begin{aligned} x &= \frac{t_b - t_a + r \cdot L \cdot (C_{T_b} + c \cdot L/2)}{r \cdot L \cdot (c \cdot L + C_{T_a} + C_{T_b})} \\ &= \frac{0.96 - 13.44 + 0.1 \cdot 10 \cdot (5 + 1)}{0.1 \cdot 10 \cdot (2 + 30 + 5)} \\ &= -0.175 < 0 \end{aligned}$$

As $x < 0$, it is necessary to use a wire length longer than $L = 10$ units to balance the delays t_a and t_b. We assign $|e_a| = 0$ and solve for $|e_b|$ as follows:

$$\begin{aligned} t_a &= t_b + r \cdot |e_b| \cdot \left(C_{T_b} + \frac{c \cdot |e_b|}{2} \right) \\ \Rightarrow 13.44 &= 0.96 + 0.1 \cdot |e_b| \cdot \left(5 + \frac{0.2 \cdot |e_b|}{2} \right) \\ \Rightarrow |e_b| &= 18.28 \text{ units} \end{aligned}$$

At this point, we have the total wire length of the zero-skew routing tree, which is $|e_{s_1}| + |e_{s_2}| + |e_{s_3}| + |e_{s_4}| + |e_a| + |e_b| = 8 + 12 + 6 + 4 + 0 + 18.28 = 48.28$ units.

The merging segment ms_v is obtained by taking the intersection of trr_a and trr_b, as shown in Figure 13.29. As $|e_a| = 0$, trr_a is simply ms_a. The tilted rectangular region trr_b, whose vertices are $(-17.28, 15)$, $(1, 33.28)$, $(23.28, 11)$ and $(5, -7.28)$, overlaps

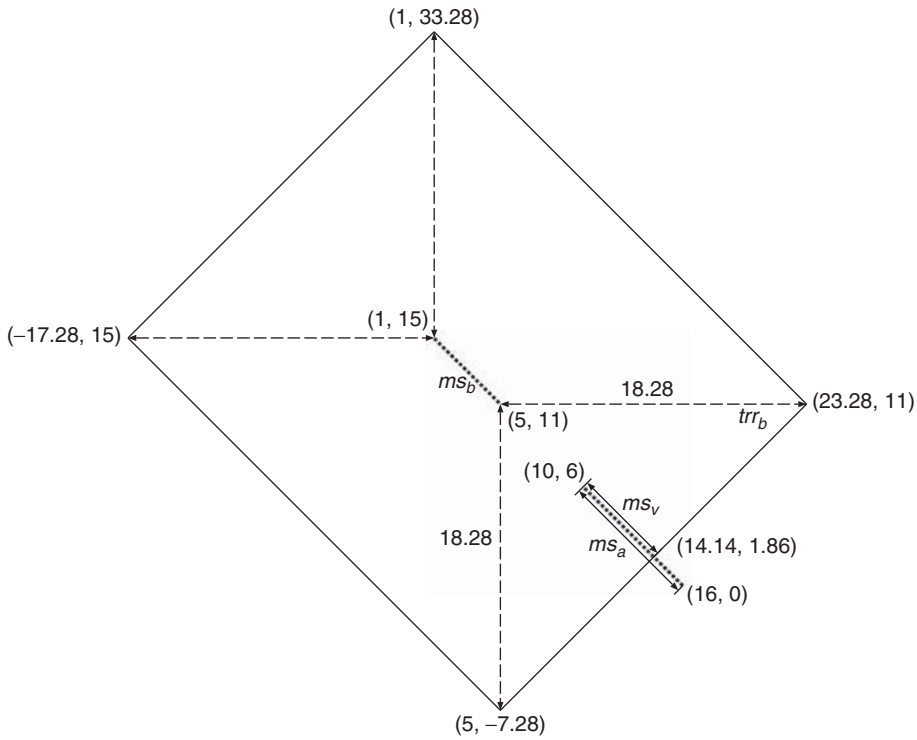


FIGURE 13.29

Computation of ms_v.

with $ms_a = trr_a$. Consequently, the two endpoints of ms_v are $(10, 6)$ and $(14.14, 1.86)$, as shown in Figure 13.29.

Now, let us turn our attention to the top-down embedding of the internal nodes of the topology G. Because the location of clock source $s_0 = v$ is not given, we have the flexibility of embedding $s_0 = v$ at any point on ms_v. Let us consider the two endpoints of ms_v for embedding:

1. We embed $s_0 = v$ at location $(10, 6)$. As $|e_a| = 0$, we also have to embed a at the same location. As $|e_b| = 18.18 > d(l_v, ms_b) = 10$, the snaking of wire may be required to balance the delay. To determine l_b, we use the intersection of ms_b and the tilted rectangular region bounded by a Manhattan circle whose center is $l_v = (10, 6)$ and radius is $|e_b| = 18.18$ units. As shown in Figure 13.30, we can embed b at any point on ms_b,

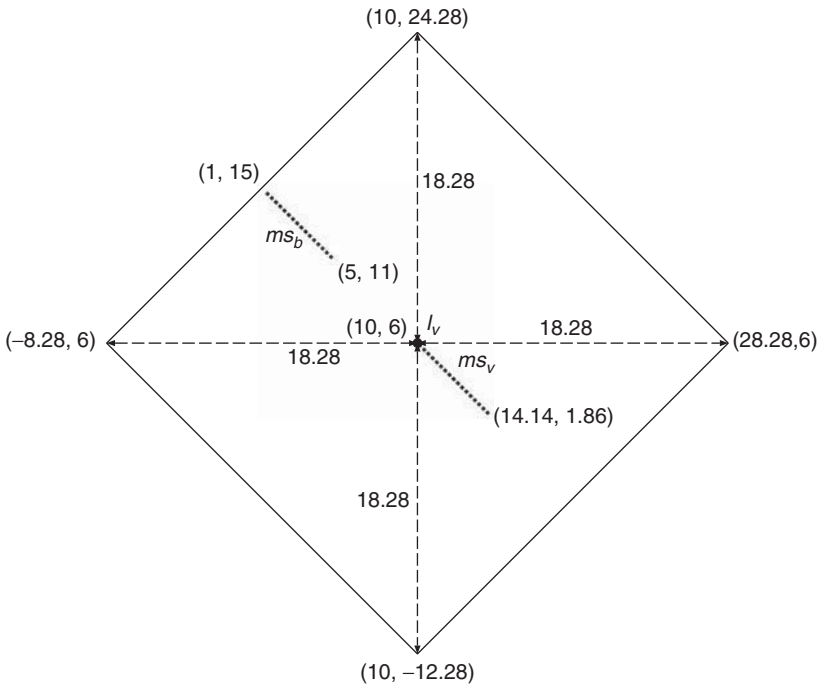


FIGURE 13.30

Embedding of internal node b.

because ms_b is completely contained within the tilted rectangular region whose vertices are $(-8.28, 6)$, $(10, 24.28)$, $(28.28, 6)$, and $(10, -12.28)$.

In Figure 13.31a, we embed b at location $(5, 11)$, which is of distance 10 units from l_v. Consequently, we require a detour of length 8.28 units in the connection from l_v to l_b. In Figure 13.31b, we embed b at location $(1, 15)$, which is of distance 18 units from l_v. As a result, the connection between l_v and l_b requires a snaking of length 0.28 units.

2. We embed $s_0 = v$ at location $(14.14, 1.86)$; that leaves the endpoint $(5, 11)$ of ms_b as the only possible location for the embedding of b. Again, we have to embed a at the same location as v. The corresponding routing tree is shown in Figure 13.31c.

It is straightforward to verify that all three zero-skew routing trees in Figure 13.31 have the same wiring cost of 48.28 units.

Because DME requires an input topology, the generation of a good abstract topology is crucial. In fact, many of the more successful approaches interleave topology construction with merging segment computation. The Greedy-DME

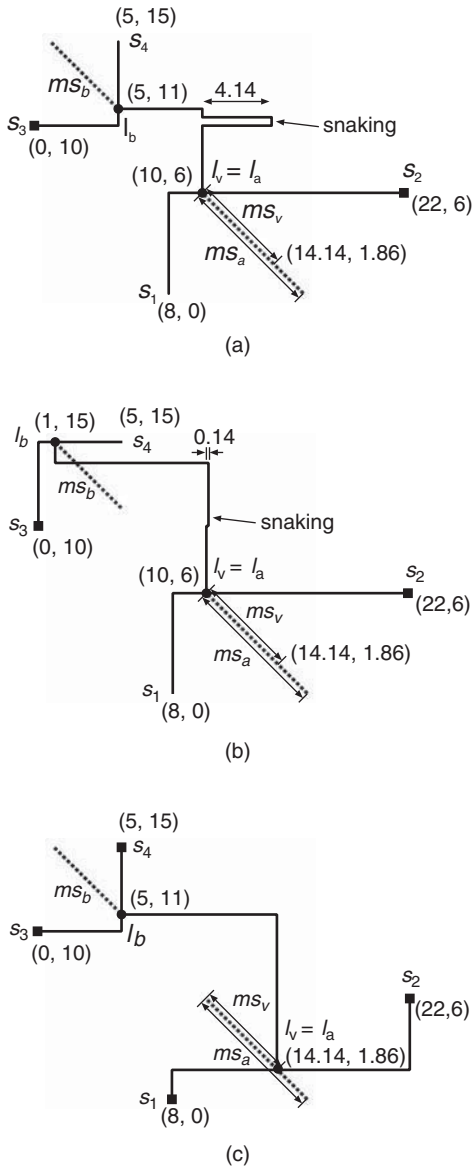


FIGURE 13.31

Three different ways of embedding the merging tree in Figure 13.27 with the same wiring cost. The embedded routing trees in (a) and (b) require detour wirings, whereas the embedding in (c) does not.

method proposed in [Edahiro 1992] is the most successful among them. Let F denote a forest of singleton merging trees, each consisting of only a single sink location. Greedy-DME iteratively finds the "nearest" pair of neighbors in F, say ms_a and ms_b, and constructs for the newly added parent node, say v, a merging segment ms_v based on a zero-skew merge of ms_a and ms_b. To account for detour wiring, the proximity of two merging segments is usually defined by the cost of merging them instead of their physical distance. The merging trees rooted at ms_a and ms_b in F are then replaced by a new merging tree rooted at ms_v. After $n - 1$ merging operations, F contains a single merging tree, which also corresponds to the abstract topology of the zero-skew routing tree. The outline of Greedy-DME is similar to that of DME except for step 2, which is amended as below:

---

2. **for** each merging of two subtrees T_a and T_b that are nearest neighbors in F to form T_v **do**

In the Greedy-DME algorithm, it takes $O(n)$ iterations to construct a zero-skew clock tree, with each iteration involving an $O(n^2)$ procedure to identify the nearest-neighbor pair for merging. We can reduce the number of iterations to $O(\log n)$ by merging several nearest-neighbor pairs simultaneously [Edahiro 1993a]. In each iteration, we construct a "nearest-neighbor graph" that maintains the nearest neighbor of each merging tree in F. In nondecreasing order of distance, $|F|/k$ ($2 \leq k \leq 4$) independent nearest-neighbor pairs of merging trees are chosen from the graph for zero-skew merging. The number of merging trees in F is reduced by a factor of $1/k$ after each iteration. Consequently, it takes only $O(\log_{k/(k-1)} n)$ iterations to construct a zero-skew routing tree.

The construction of a nearest-neighbor graph in each iteration has a quadratic time complexity. An approximate nearest-neighbor graph can be constructed in linear time by use of the bucket decomposition method in [Edahiro 1994]. In each iteration, the smallest square routing plane that covers all merging segments of the root nodes of the merging trees in F is uniformly partitioned into $\Theta(|F|)$ square buckets. The routing plane and buckets are all tilted by 45 degrees, as shown in Figure 13.32. Each bucket has at most eight neighboring buckets. Assuming a uniform distribution of merging segments, the number of merging segments in each bucket is $O(1)$. We restrict the nearest neighbor of a merging segment, say ms_v, to reside within the same bucket(s) as ms_v or in the adjacent buckets as shown in Figure 13.32. Consequently, an approximate nearest-neighbor graph can be constructed in linear time.

13.3.3.4 *Bounded-skew routing*

The two-phase approach taken by the DME algorithm to compute a zero-skew tree for a prescribed topology can be extended quite naturally to handle more general skew constraints. The BST/DME solutions developed in [Cong 1995a; Huang 1995; Cong 1995b] for the problem of constructing a bounded-skew

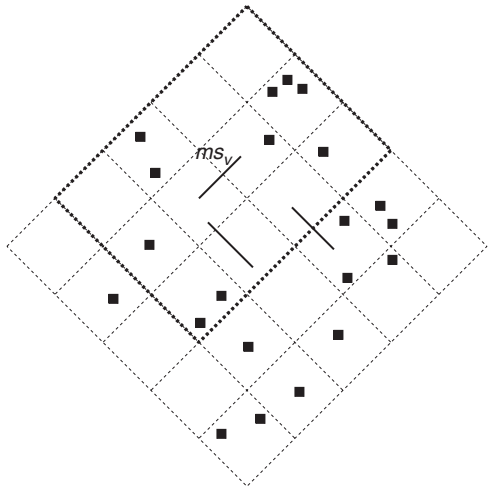


FIGURE 13.32

Bucket decomposition of a routing plane.

routing tree generalize the concept of a merging segment for zero-skew routing to that of a *merging region*. In a BST/DME algorithm, a merging region contains all candidate locations of the corresponding internal node in the bounded-skew tree. The bottom-up process constructs a tree of merging regions (in contrast to merging segments for zero-skew tree), and the top-down process then determines the exact locations of all internal nodes. Figure 13.33 shows the merging regions and routing tree constructed by BST/DME for the example in Figure 13.23 (and Figure 13.26).

The construction of the merging region for the parent node, say v, of two sinks, say s_i and s_j, is quite straightforward. First, we construct a bounded-skew merging segment ms_v^+ such that $t_i^+ = t_j^+ + B$, where t_i^+ and t_j^+ are, respectively, the delays from the merging segment ms_v^+ to s_i and s_j, and B is the skew bound. The computation of this bounded-skew merging segment is similar to that of a zero-skew merging segment. Let $L = d(l_{s_i}, l_{s_j})$ denote the shortest Manhattan distance between l_{s_i} and l_{s_j}, and $x^+ \cdot L$ the distance between ms_v^+ and l_{s_i}, where x^+ is between 0 and 1 if detour wiring is not necessary. Then, we solve for x^+ from the following expression:

$$\underbrace{r \cdot x^+ \cdot L \cdot \left(c_{s_1}^g + \frac{c \cdot x^+ \cdot L}{2} \right)}_{t_j^+} = \underbrace{r \cdot (1 - x^+) \cdot L \cdot \left(c_{s_j}^g + \frac{c \cdot (1 - x^+) \cdot L}{2} \right)}_{t_j^+} + B$$

where $c_{s_i}^g$ and $c_{s_j}^g$ are, respectively, the gate capacitances of the clock sinks s_i and s_j. For simplicity, we first consider the case that x^+ obtained from the preceding expression is, indeed, between 0 and 1. It is obvious that a tree that uses

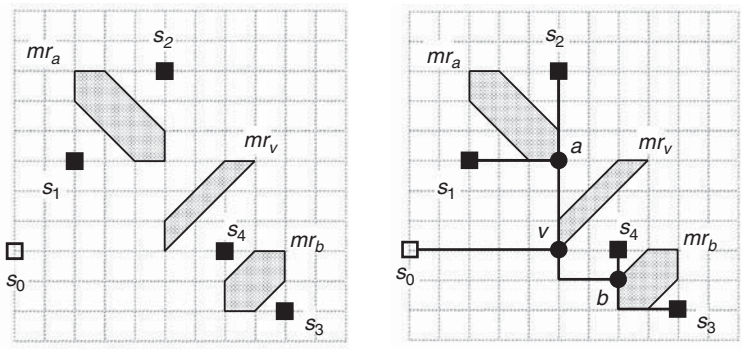


FIGURE 13.33

(a) Merging regions constructed by BST/DME for the example in Figure 13.23. (b) Embedding of internal nodes. The bounded-skew routing has a lower routing cost than the zero-skew routing in Figure 13.26.

shortest wire lengths to connect any point on ms_v^+ to s_i and s_j would have a maximum sink delay of t_i^+ and a minimum sink delay of t_j^+, satisfying the skew bound constraint B.

Similarly, we construct a bounded-skew merging segment ms_v^- such that $t_i^- = t_j^- - B$, where t_i^- and t_j^- are, respectively, the delays from the merging segment ms_v^- to s_i and s_j. The bounded-skew merging segment ms_v^- should be at a distance of $x^- \cdot L$ from l_{s_i}. Again, assume that x^- is between 0 and 1. A tree that uses shortest wire lengths to connect any point on ms_v^- to s_i and s_j would have a minimum sink delay of t_i^- and a maximum sink delay of t_j^-. The region bounded by ms_v^+ and ms_v^- within the smallest bounding box containing s_i and s_j is the merging region mr_v.

In Figure 13.33, two merging regions mr_a and mr_b are obtained by merging clock sinks under the path length delay model. One can easily verify that the bounded-skew merging segments are simply the corresponding zero-skew merging segments shifted toward or away from the corresponding clock sink.

A few important properties are associated with a merging region, say mr_v, whose two child nodes are clock sinks. By construction, $x^- \leq x^+$. Now, consider a Manhattan arc in mr_v that is of distance between x^- and x^+ from s_i, pick any point residing on this Manhattan arc, and construct a routing tree that connects this point to s_i and s_j, with the shortest wire lengths. It is fairly straightforward to verify that such a routing tree would satisfy the skew-bound constraint B. Moreover, all routing trees constructed in a similar manner with root nodes residing on this Manhattan arc would have the same maximum sink delay and the same

minimum sink delay. Furthermore, $|e_{s_i}| + |e_{s_j}| = d(l_{s_i}, l_{s_j})$ regardless of where v is embedded within mr_v. Consequently, $C_{T_v} = c_{s_i}^g + c_{s_j}^g + c \cdot d(l_{s_i}, l_{s_j})$.

Now, let us consider the more complicated cases when x^+ or x^- may not be between 0 and 1:

$0 \le x^- \le 1 < x^+$: We force ms_v^+ to be at a distance $L = d(l_{s_i} - l_{s_j})$ from s_i. In other words, ms_v^+ coincides with l_{s_j}.

$x^- < 0$ and $1 < x^+$: We force ms_v^- and ms_v^+ to be, respectively, at distances of 0 and L from s_i. In other words, ms_v^- coincides with l_{s_i} and ms_v^+ coincides with l_{s_j}.

$x^- < 0 \le x^+ \le 1$: We force ms_v^- to be at a distance of 0 from s_i. In other words, ms_v^- coincides with l_{s_i}.

In other words, we always force x^- and x^+ to be between 0 and 1. Note that as we are merging two clock sinks, it is not possible for x^- and x^+ to be both less than 0 or greater than 1. Such is not the case when we construct merging regions for internal nodes higher up in the abstract topology.

Let two non-leaf nodes a and b be the child nodes of v. Given merging regions mr_a and mr_b, we follow the approach of boundary merging and embedding in [Cong 1995a; Huang 1995], where mr_v is constructed from the nearest *boundary segments* of mr_a and mr_b. A point p on the nearest boundary segment of mr_a, called a *joining segment* and denoted JS_a, can merge with a point q on joining segment JS_b if $d(p, q) = d(mr_a, mr_b)$.

Suppose mr_a and mr_b are also octilinear convex polygons as shown in Figure 13.33. Therefore, the joining segments from mr_a and mr_b are either parallel Manhattan arcs or parallel rectilinear line segments. We will now present the merging of two joining segments that are Manhattan arcs. As pointed out earlier, all bounded-skew routing trees that have their root nodes embedded on JS_a have the same maximum sink delay and the same minimum delay. Let $t_a^{\max}$ and $t_a^{\min}$ denote such delays, respectively. Similarly, $t_b^{\max}$ and $t_b^{\min}$ are defined for JS_b.

Let $L = d(JS_a, JS_b) = d(mr_a, mr_b)$ denote the distance between JS_a and JS_b. We compute a merging segment ms_v^+ such that in a routing tree obtained by making the shortest connections from a point on ms_v^+ to JS_a and JS_b, the maximum and minimum sink delays in the tree are, respectively, caused by sinks in T_a and T_b, and the skew is no greater than B. The distance of ms_v^+ from JS_a can be obtained by solving for x^+ in the following equation:

$$t_a^{\max} + r \cdot x^+ \cdot L \cdot \left(C_{T_a} + \frac{c \cdot x^+ \cdot L}{2} \right)$$
$$= t_b^{\min} + r \cdot (1 - x^+) \cdot L \cdot \left(C_{T_b} + \frac{c \cdot (1 - x^+) \cdot L}{2} \right) + B$$

If x^+ is between 0 and 1, ms_v^+ is at a distance of $x^+ \cdot L$ from JS_a. Similarly, we compute the location of a merging segment ms_v^- by solving for x^- in the following equation:

$$t_a^{\min} + r \cdot x^- \cdot L \cdot \left(C_{T_a} + \frac{c \cdot x^- \cdot L}{2} \right)$$
$$= t_b^{\max} + r \cdot (1 - x^-) \cdot L \cdot \left(C_{T_b} + \frac{c \cdot (1 - x^-) \cdot L}{2} \right) - B$$

If x^- is between 0 and 1, ms_v^- is at a distance of $x^- \cdot L$ from JS_a.

Suppose x^- and x^+ are both between 0 and 1, the region bounded by ms_v^+ and ms_v^- within the smallest bounding box containing JS_a and JS_b is the merging region mr_v, as shown in Figure 13.34. The merging region mr_v is constructed under the path length delay model. Each Manhattan arc is associated with a pair of numbers, the maximum and minimum sink delays from a point on that arc. The maximum (minimum) sink delay of ms_v^+ is due to some sink in T_a (T_b), whereas the maximum (minimum) sink delay of ms_v^- is due to some sink in T_b (T_a).

Similar to the merging of two clock sinks, so long as x^- and x^+ are not both less than 0 or greater than 1, we would always force x^- and x^+ to be between 0 and 1. When x^- and x^+ are both less than 0 or when they are both greater than 1, detour wiring is necessary. In the former, we set $|e_a| = 0$, make mr_v coincide with JS_a, and solve for $|e_b|$ in the following equation:

$$t_a^{\max} = t_b^{\min} + r \cdot |e_b| \cdot \left(C_{T_b} + \frac{c \cdot |e_b|}{2} \right) + B$$

In the latter, we set $|e_b| = 0$, make mr_v coincide with JS_b, and solve for $|e_a|$ in the following equation:

$$t_a^{\min} + r \cdot |e_a| \cdot \left(C_{T_a} + \frac{c \cdot |e_a|}{2} \right) = t_b^{\max} - B$$

However, detour wiring may not be necessary after all. Take the case in which x^- and x^+ are both less than 0, the tilted rectangular region defined by JS_b and

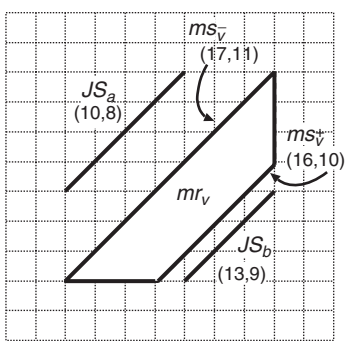


FIGURE 13.34

Merging of two Manhattan joining segments JS_a and JS_b for a skew bound of 6. The numbers associated with each Manhattan arc are the maximum and minimum sink delays from a point on that arc.

$|e_b|$ actually covers more than JS_a; it actually overlaps with mr_a. That implies that it may actually be more economical to use interior points of mr_a, instead of boundary segments, for merging. Another reason for the use of interior points is when mr_a and mr_b overlap. However, merging of interior points presents tremendous challenges, because every point in a merging region may correspond to the merging of many different pairs of interior points from its child merging regions.

To overcome the difficulty in the use of interior points for merging, we introduce the notion of *sampling segments* as in [Cong 1995b]. A merging region is represented by a set of s sampling segments, each of which is a Manhattan arc that is parallel to the 45 degrees boundary segments of the merging region. Such a sampling segment has the property that it has a constant maximum sink delay and a constant minimum sink delay on any point along the segment. Figure 13.35 illustrates the concept of sampling segments under the path length delay model. The maximum and minimum sink delays associated with each sampling segment are also shown in parentheses.

Merging of two nodes now involves two sets of sampling segments. Clearly, we can apply the approach for merging two joining segments outlined earlier to accomplish the merging of two sampling segments from two merging regions. The challenge here is that the approach would generate a set of up to s^2 merging regions for the parent node. If each of these merging regions is, in turn, sampled by s segments, the time and space complexity of this approach would grow exponentially. For an efficient and practical implementation of such an

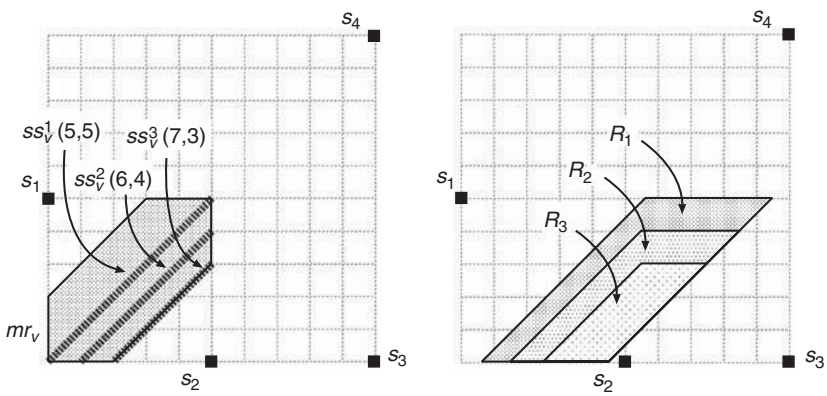


FIGURE 13.35

(a) Sampling of mr_v (obtained from the merging of s_1 and s_2) with $\{ss_v^1, ss_v^2, ss_v^3\}$. (b) Merging these sampling segments with sink s_3 with a skew bound of 2 units produces three merging regions where R_i is produced by merging s_3 with ss_v^i.

approach, the number of merging regions associated with a node is typically limited to a constant, say k. Each region is, in turn, sampled by s sampling segments. Therefore, the merging of two nodes will generate k^2s^2 merging regions. One typical approach to pruning these k^2s^2 merging regions is to keep only the best k merging regions, defined in terms of merging cost, for the parent node.

Merging with sampling segments facilitates merging of interior points. Because 45 degrees joining segments are on the boundary of a merging region, the skew associated with each joining segment is in general higher than the skew associated with other sampling segments from the same merging region. Consequently, merging of interior sampling segments may result in a larger merging region at a parent node. In Figure 13.35, R_3 is also the merging region produced by merging mr_v with s_3 by use of a boundary joining segment. In this example, R_3 is smaller than R_1 and R_2, both of which are constructed with interior sampling segments. The larger R_1 and R_2 may lead to a reduced wiring cost at the next merging step.

The generation of the k^2s^2 merging region for each node in an abstract topology may not sound efficient. However, another benefit of the use of sampling segments in the merging process is that the merging of joining segments that are horizontal or vertical is obviated. Under the Elmore delay model, the merging of two parallel horizontal (or vertical) joining segments results in a merging region that is no longer octilinear. Consequently, subsequent merging operations may involve joining segments that are neither rectilinear line segments nor Manhattan arcs. Moreover, the computation of such a merging region requires $O(n)$ runtime and space complexity for a subtree that contains n sinks [Cong 1995b].

Example 13.3 Bounded-Skew Routing under Elmore Delay Model: We again consider the 4-sink example in Figure 13.27. Instead of zero-skew routing, we allow a skew bound of 2.5 ns (*i.e.*, $B = 2.5$ ns). First, we compute x^+ and x^- for the merging of s_1 and s_2, where $x^+ \cdot d(l_{s_1}, l_{s_2})$ and $x^- \cdot d(l_{s_1}, l_{s_2})$, respectively, denote the distances of ms_a^+ and ms_a^- from l_{s_1}:

$$
\begin{aligned}
x^+ &= \frac{B + r \cdot L \cdot (C_{T_{s_2}} + c \cdot L/2)}{r \cdot L \cdot (c \cdot L + C_{T_{s_1}} + C_{T_{s_2}})} \\
&= \frac{2.5 + 2 \cdot (10 + 2)}{2 \cdot (4 + 16 + 10)} \\
&= 0.44 \\
x^- &= \frac{-B + r \cdot L \cdot (C_{T_{s_2}} + c \cdot L/2)}{r \cdot L \cdot (c \cdot L + C_{T_{s_1}} + C_{T_{s_2}})} \\
&= \frac{-2.5 + 2 \cdot (10 + 2)}{2 \cdot (4 + 16 + 10)} \\
&= 0.36
\end{aligned}
$$

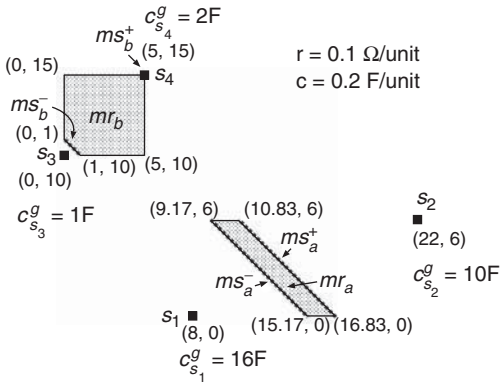


FIGURE 13.36

Merging regions of internal nodes a and b.

where $L = d(l_{s_1}, l_{s_2}) = 20$ units. We show the merging segments ms_a^+ and ms_a^- in Figure 13.36. The shaded region bounded by ms_a^+ and ms_a^- is the merging region ms_a, whose vertices are $(9.17, 6)$, $(10.83, 6)$, $(16.83, 0)$, and $(15.17, 0)$.

To compute the merging region of b, we define $x^+ \cdot d(l_{s_3}, l_{s_4})$ and $x^- \cdot d(l_{s_3}, l_{s_4})$ to be the distances of ms_b^+ and ms_b^- from l_{s_3}, respectively. We obtain $x^+ = 1.1$ and $x^- = 0.1$. Because $x^+ > 1$, we force $x^+ = 1$. In other words, ms_b^+ coincides with l_{s_4}. We also show the merging segments and merging region of b in Figure 13.36.

Now, we consider two different ways to merge a and b. First, we consider the case of merging a and b with joining segments from mr_a and mr_b that are closest (*i.e.*, $d(JS_a, JS_b) = d(mr_a, mr_b) = 8.17$ units). In this case, JS_a is simply the point $(9.17, 6)$ and JS_b is $(5, 10)$. For JS_a, we have the following parameters: $t_a^{\max} = t_2 = 14.48$ ns, $t_a^{\min} = t_1 = 11.98$ ns, and $C_{T_a} = 30$ F. For JS_b, we have $t_b^{\max} = t_4 = 1.25$ ns, $t_b^{\min} = t_3 = 0.75$ ns, and $C_{T_b} = 5$ F. Letting $L = d(JS_a, JS_b)$ and defining $x^+ \cdot L$ and $x^- \cdot L$ to be the distances of ms_v^+ and ms_v^- from JS_a, respectively, we obtain x^+ and x^- as follows:

$$
\begin{aligned}
x^+ &= \frac{t_b^{\min} - t_a^{\max} + B + r \cdot L \cdot (C_{T_b} + c \cdot L/2)}{r \cdot L \cdot (c \cdot L + C_{T_a} + C_{T_b})} \\
&= \frac{0.75 - 14.48 + 2.5 + 0.817 \cdot (5 + 0.817)}{0.817 \cdot (1.63 + 30 + 5)} \\
&= -0.22, \\
x^- &= \frac{t_b^{\max} - t_a^{\min} - B + r \cdot L \cdot (C_{T_b} + c \cdot L/2)}{r \cdot L \cdot (c \cdot L + C_{T_a} + C_{T_b})} \\
&= \frac{1.25 - 11.98 + 2.5 + 0.817 \cdot (5 + 0.817)}{0.817 \cdot (1.63 + 30 + 5)} \\
&= -0.28
\end{aligned}
$$

Because both x^+ and x^- are negative, we force $|e_a| = 0$ (*i.e.,* ms_v is JS_a). Then, we compute $|e_b|$ with the following equation:

$$t_a^{\max} = t_b^{\min} + r \cdot |e_b| \cdot \left(C_{T_b} + \frac{c \cdot |e_b|}{2} \right) + B$$

$$\Rightarrow 14.48 = 0.75 + 0.1 \cdot |e_b| \cdot \left(5 + \frac{0.2 \cdot |e_b|}{2} \right) + 2.5$$

$$\Rightarrow |e_b| = 16.81 \text{ units}$$

Because $d(JS_a, JS_b) = 8.17$ units, a detour wiring of length 8.64 units is required. Figure 13.37 shows the bounded-skew routing tree constructed accordingly. The total wire length of the routing tree is 46.81 units.

Now, we consider the case of merging a and b with sampling segments from mr_a and mr_b. For any merging region mr, we always include as sampling segments the merging segments ms^+ and ms^-, which are Manhattan arcs that define the boundary segments of mr. In Figure 13.38, which shows the sampling of mr_a and mr_b, it is clear that ms_a^+,

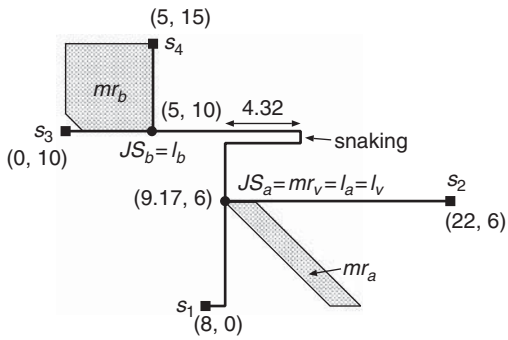


FIGURE 13.37

A bounded skew routing tree constructed based on the merging of the nearest boundary segments.

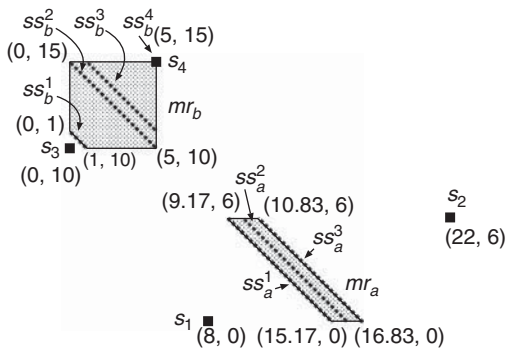


FIGURE 13.38

Sampling of the merging regions mr_a and mr_b.

ms_a^-, ms_b^+, and ms_b^- have been included as sampling segments ss_a^1, ss_a^3, ss_b^1, and ss_b^4, respectively. Each merging region has a subregion whose clock skew among its sinks is the smallest. Sampling of such a subregion is desirable, because it usually leads to a larger merging region at the parent node. In this example, both mr_a and mr_b contain the zero-skew merging segments, which are included as sampling segments ss_a^2 and ss_b^3. We also include sampling segments that contain the joining segments. Consequently, ss_b^2 is a sampling segment of mr_b. It is important to note that all sampling segments from a single merging region have the same wiring cost.

Now, we perform the pairwise merging of ss_a^i and ss_b^j, for $i \in \{1,2,3\}$ and $j \in \{1,2,3,4\}$. It turns out that in every pairwise merging, $|e_a| = 0$ and detour wiring in e_b is required. Table 13.1 shows the different combinations of $|e_b|$, $t_v^{\max}$, and $t_v^{\min}$ obtained by these pairwise merging operations. It should be evident from Table 13.1 that the $t_v^{\max}$ and $t_v^{\min}$ values satisfy the skew-bound constraint of 2.5 ns in all cases.

Because e_b requires detour wiring, the merging regions obtained in these pairwise mergings overlap with the sampling segments ss_a^i. Each of these merging regions (or regions that degenerate into segments) is obtained by taking the intersection of trr_b (constructed from the respective ss_b^j and $|e_a|$) and ss_a^i (see Section 13.3.3.3). The coordinates of the merging regions of v are given in Table 13.2. It is important to note that

Table 13.1 Wiring Costs and Sink Delays Obtained by Pairwise Merging of Sampling Segments from mr_a and mr_b.

	ss_a^1			ss_a^2			ss_a^3		
	$\|e_b\|$	$t_v^{\max}$ (ns)	$t_v^{\min}$ (ns)	$\|e_b\|$	$t_v^{\max}$ (ns)	$t_v^{\min}$ (ns)	$\|e_b\|$	$t_v^{\max}$ (ns)	$t_v^{\min}$ (ns)
ss_b^1	17.57	14.48	11.98	16.33	13.44	10.94	18.07	14.91	12.41
ss_b^2	16.81	14.48	11.98	15.55	13.44	10.94	17.32	14.91	12.41
ss_b^3	16.56	14.48	11.98	15.29	13.44	10.94	17.08	14.91	12.41
ss_b^4	17.70	14.48	11.98	16.46	13.44	10.94	18.20	14.91	12.41

Table 13.2 Merging Regions Obtained by Pairwise Merging of Sampling Segments from mr_a and mr_b.

	ss_a^1	ss_b^2	ss_b^3
ss_b^1	(9.17,6)–(11.87,3.30)	(10,6)–(11.66,4.34)	(10.83,6)–(12.95,3.88)
ss_b^2	(9.17,6)–(13.49,1.68)	(10,6)–(13.27,2.73)	(10.83,6)–(14.58,2.25)
ss_b^3	(9.17,6)–(12.86,2.30)	(10,6)–(12.64,3.36)	(10.83,6)–(13.95,2.88)
ss_b^4	(9.17,6)–(11.43,3.73)	(10,6)–(11.23,4.77)	(10.83,6)–(12.52,4.32)

only when detour wiring occurs does the computation of a degenerate merging region from sampling segments resemble the computation of a zero-skew merging segment. Recall that detour wiring is required for bounded-skew merging only when x^+ and x^- are both greater than 1 or both less than 0. Figure 13.39 shows the construction of 4 merging regions of v by merging ss_a^2 with all 4 sampling segments of mr_b. For ease of reference, we refer to the respective tilted rectangular region of sampling segment ss_b^j as trr_b^j and the resultant merging region as mr_v^j in the illustration.

Among these, the merging of ss_a^2 and ss_b^3 results in the lowest $|e_b|$ and, hence, a minimum-cost bounded-skew tree. Figure 13.40 shows the results of embedding v at two different locations in the merging region mr_v^3 obtained from the merging of ss_a^2 and ss_b^3.

In Figure 13.40a, v is embedded at $(12.64, 3.36)$, an endpoint of the merging region mr_v^3. Consequently, that leaves $(5, 11)$ as the only possible location for the embedding of b. When we embed v at $(10, 6)$, the possible locations for the embedding of b lie on the Manhattan arc defined by $(2.36, 13.64)$ and $(5, 11)$. Figures 13.40b,c show the results of embedding b at these two endpoints. As the embedding location of b moves from $(2.36, 13.64)$ to $(5, 11)$, the amount of snaking increases. All three embedded routing trees have the same total wiring cost of 45.29 units, which is lower than those of the bounded-skew routing tree in Figure 13.37 and the zero-skew routing trees in Figure 13.31.

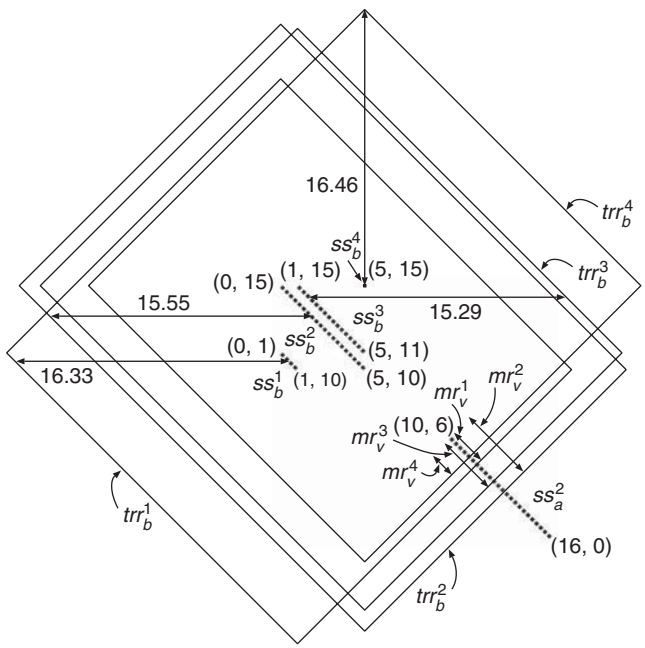


FIGURE 13.39

Construction of merging regions of v by merging ss_a^2 with all 4 sampling segments of mr_b.

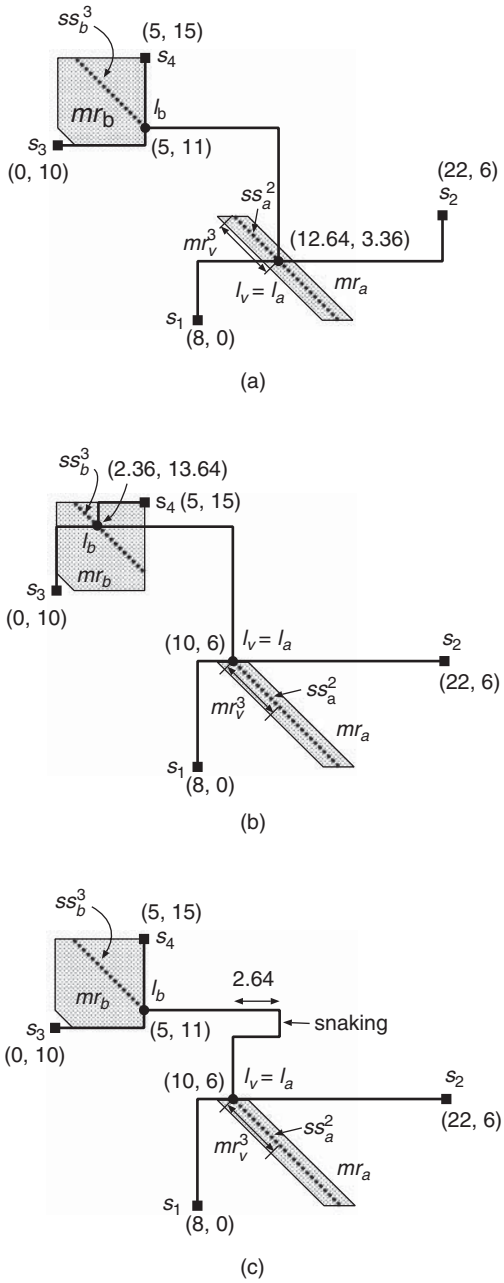


FIGURE 13.40

Bounded-skew routing trees constructed based on the merging of sampling segments.

Suppose v is not the clock source. It may be necessary to keep only a subset of the 12 merging regions for the construction of merging regions of the parent node of v. If we allow each node to be associated with only 3 merging regions, mr_v^1, mr_v^2, and mr_v^3 would be chosen on the basis of their lower merging costs. Although these merging regions overlap, it is important to note that they would now have different C_{T_v}'s, which could play an important role in determining the merging regions (and their associated merging costs) of the parent node.

For the generation of an abstract topology, we can take an approach similar to that of the Greedy-DME algorithm, with various acceleration methods incorporated. Indeed, the Greedy-BST/DME algorithm in [Huang 1995] is very similar to the Greedy-DME algorithm. However, the Greedy-BST/DME algorithm has the added flexibility that it allows merging of two subtrees at non-root nodes, whereas Greedy-DME always merges two subtrees at their root nodes. Merging with non-root nodes is an effective topology optimization method. In fact, it very closely matches the performance of the best-known heuristics for both the zero-skew and infinite-skew limiting cases (*i.e.,* Steiner routing).

Consider, for example, a clock network with eight sinks that are equally spaced on a horizontal line, as shown in Figure 13.41. Although the embedding is not shown here, it can be easily verified that the abstract topology to the left, which is obtained by the merging of T_1 and T_2 at their root nodes, can be embedded to produce a minimum-cost zero-skew tree. Although T_1 and T_2 are themselves ideal topologies for low-cost embedding for large skew bound B, the merging of them at their root nodes is quite costly (see the embedding following the abstract topology). The reason is that the root nodes (embedded at locations labeled "3" and "6," which correspond to the locations of sinks s_3 and s_6) are quite far apart. If we adjust the subtree topology such that the roots of subtrees become closer (see the abstract topology and embedding to the right), the overall tree cost would be reduced. Here, the root nodes of the adjusted topologies T_1 and T_2 are embedded at locations labeled "4" and "5" (or l_{s_4} and l_{s_5}).

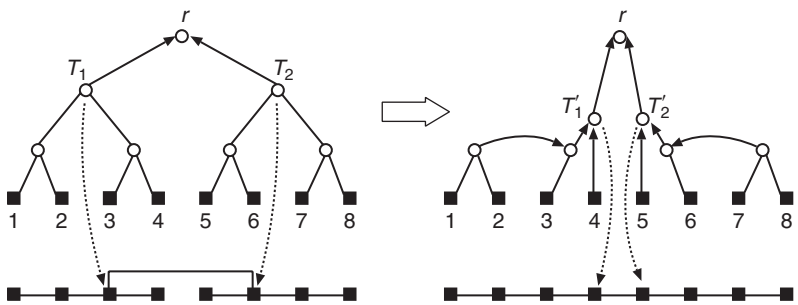


FIGURE 13.41

An example showing that given skew bound $B \gg 0$, changing the subtree topology before merging will reduce the merging cost.

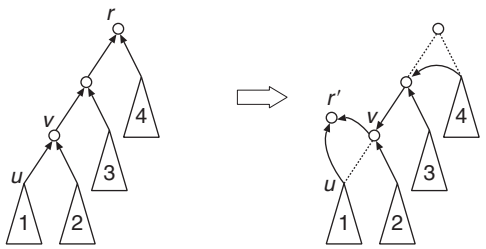


FIGURE 13.42

Repositioning the root in changing the topology.

Figure 13.42 illustrates in more detail how the tree topology is adjusted. To move the root node r to some tree edge, say $e_u = uv$, we traverse along the path from r to v. For each edge encountered along the top-down path traversal, we delete the edge and merge the two appropriate subtrees off the deleted edge. To remove the left edge of r, for example, we merge subtrees labeled "3" and "4." This newly merged tree of "3" and "4" is then combined with the subtree labeled "2" for the deletion of the parent edge of v. This tree is then merged with the subtree labeled "1" such that the new root node r' breaks the edge uv. Effectively, we have relocated the root node to be closer to the subtree labeled "1."

The re-rooting approach relocates r to uv by remerging appropriate subtrees along the path from r to v. Hence, we also know the merging region corresponding to the bounded-skew tree rooted at the parent edge of v. Indeed, a careful examination of the approach would reveal that a simple $O(n)$ top-down traversal of the abstract topology rooted at r would compute all merging regions corresponding to the relocation of the root node to all tree edges in the topology.

To incorporate such a feature in the Greedy-BST/DME algorithm, each node v in an abstract topology G (in the forest F) is associated with two merging regions, denoted as mr_v and mr_{e_v}. The former merging region mr_v is the merging region constructed for v on the basis of the sub-topology rooted at v in G. The latter merging region mr_{e_v} is the merging region if the root node of G is relocated to the parent edge of v in G (*i.e.,* e_v).

To identify in F the nearest neighbor of G, whose root node is r, we consider mr_r for the root node, as well as mr_{e_v} for all other nodes v in G. In other words, some other topologies in F, possibly rerooted as well, may be closer to G after G is rerooted. In the example given in Figure 13.41, the nearest neighbors of the sub-topologies T_1 and T_2 are $mr_{e_{s_4}}$ and $mr_{e_{s_5}}$.

Consequently, the construction of the nearest neighbor graph in Greedy-BST/DME always involves n nodes, because all nodes in all the sub-topologies in F participate in the identification of the nearest neighbors. We will examine the complexity of the Greedy-BST/DME algorithm in one of the exercises.

13.3.3.5 *Useful-skew routing*

The useful-skew routing problem refers to the problem of synthesizing a clock routing that satisfies all specified general clock skew constraints (*i.e.,* setup and hold-time constraints). There is a variant of the useful-skew routing problem that deals with a prescribed clock schedule (*i.e.,* all skew constraints are equality constraints instead of being bounded from above and below). Such a variant can be solved by a simple modification of the zero-skew routing algorithm. In the sequel, we deal with the more general problem where we maintain the flexibility of skew scheduling throughout the process of constructing a useful-skew clock routing tree [Xi 1996; Taso 2002].

In Section 13.3.3.1, we have already introduced some concepts that would be of crucial importance to useful-skew routing, namely, the feasible-skew range for a pair of clock sinks, the commitment of the skew of a pair of clock sinks, and the incremental updates of the remaining feasible skew ranges after a skew commitment. Now we will show how these concepts interact with a clock tree-embedding algorithm, called UST/DME [Tsao 2002].

Because the clock skews are also constrained to lie within a range specified by some upper and lower bounds, as in the case of bounded-skew constraint in bounded-skew routing, the underlying concept of the merging region in the UST/DME algorithm is the same as that in the BST/DME algorithm. However, the interaction of merging regions and incremental scheduling introduces a problem not encountered before. In BST/DME, when a merging region of a parent node is computed from two sampling segments of the child nodes, it implies that we have committed to some maximum and minimum sink delays associated with the sampling segments. However, the commitment does not affect the skew-bound constraint, because they are being applied to all pairs of clock sinks. That is not the case in useful-skew clock routing; a skew commitment in one subtree affects the feasible skew ranges of clock pins in another.

Therefore, the first step of the UST/DME algorithm is the construction of a constraint graph G_C from the given skew constraints $\mathcal{C}$, and the second step is the computation of an all-pairs shortest distance matrix $D = \{d_{i,j} : s_i, s_j \in S\}$ from G_C to represent all feasible skew ranges $FSR_{i,j} = [-d_{i,j}, d_{j,i}]$. As before, now we consider the merging of two nodes a and b, whose parent node is v in the given abstract topology G. In other words, we construct the merging regions mr_v on the basis of mr_a and mr_b. Here, we assume that a and b are not leaf nodes. Otherwise, the merging of them is similar to the merging of two leaf nodes in BST/DME. Let the two subtrees of a be $T_{a,l}$ and $T_{a,r}$ and the two subtrees of b be $T_{b,l}$ and $T_{b,r}$.

To overcome the difficulty highlighted in the preceding paragraph (as well as the difficulties in the use of boundary segments for merging), we use a set of sampling segments to represent a merging region as in the case of BST/DME. The advantage of such a restriction is that $skew_{i,j}$ for any pair of clock pins s_i and s_j in the subtree (rooted by the sampling segment) is a constant value, say $x \in FSR_{i,j}$. We also associate each sampling segment with the leftmost clock

sink in its subtree. In other words, each sampling segment is associated with the delay to its leftmost clock sink.

Let ss_a be a sampling segment in mr_a and s_i and s_j be the representative clock pins of $T_{a,l}$ and $T_{a,r}$, respectively. Because a skew commitment of $skew_{i,j} = x$ changes $FSR_{k,b}$ for the representative clock pins s_k and s_b in $T_{b,l}$ and $T_{b,r}$, respectively, we update matrix D and recompute the merging region mr_b on the basis of the updated $FSR_{k,b}$. The updated merging region mr_b is then sampled by s sampling segments, each of which is merged with ss_a to compute a merging region mr_v for v such that the skew between s_i, the representative clock pin of T_a and s_k, the representative clock pin of T_b is feasible (*i.e.,* $skew_{i,k} \in FSR_{i,k}$). The computation of the merging region mr_v is facilitated by the associated sink delays of the two sampling segments involved.

The preceding procedure will no doubt prompt the following questions:

1. Do we have to associate a sampling segment from mr_a with the delays to all sinks in its subtree instead of just the delay to the representative clock pin?
2. Does mr_v, which satisfies the constraint $skew_{i,k} \in FSR_{i,k}$, ensure that $skew_{i',k'} \in FSR_{i',k'}$ for all $s_{i'} \in T_a$ and $s_{k'} \in T_b$ in the new tree T_v?

The answers to these questions are related. As mentioned in Section 13.3.3.1, it takes at most $n - 1$ skew commitments to the incremental skew scheduler to determine a feasible skew schedule. First, we observe that for each subtree T whose root node has a nondegenerated merging region (*i.e.,* not a Manhattan arc) there have been $|S(T)| - 2$ skew commitments made, where $S(T)$ is the set of sinks in T. When we merge two sinks, for example, we have not made a skew commitment; otherwise, we would have obtained a degenerated merging region (*i.e.,* a Manhattan arc). In other words, the subtree containing the two sinks has not made any skew commitment. The selection of a sampling segment from mr_a (for the purpose of merging) is equivalent to committing the skew between the representative clock pins of $T_{a,l}$ and $T_{a,r}$. Consequently, we have now made $|S(T_a) - 1|$ skew commitments in T_a. In other words, all skews of sink pairs in T_a have been determined. Therefore, knowing the delay from ss_a to the representative clock sink in T_a would allow us to construct all the delays to other clock sinks in T_a. Note that the merging of T_a with T_b would mean that we obtain a tree T_v with $|S(T_a)| + |S(T_b)| - 2 = |S(T_v)| - 2$ skew commitments.

To answer the second question, recall that we made an analogy between skew commitments and spanning tree construction for the complete graph representing D. Under the spanning tree analogy, one should realize that the spanning tree is built in a distributive fashion similar to that in Kruskal's algorithm [Cormen 2001]. Initially, all vertices are considered as singleton components (from the perspective of the eventual spanning tree). As skew commitments are made, connected components (which are also trees) are being joined. In each connected component, the skew between any pair of clock sinks has an equality constraint. Now, consider the merging of ss_a with ss_b (*i.e.,* the

construction of a merging region that satisfies $-d_{i,k} \leq skew_{i,k} \leq d_{k,i}$). Because $skew_{i,i'}$ is now committed for any $s_{i'} \in T_a$,

$$-d_{i,k} \leq skew_{i,k} = skew_{i,i'} + skew_{i',k} \leq d_{k,i}$$

or

$$-d_{i,k} - skew_{i,i'} \leq skew_{i',k} \leq d_{k,i} - skew_{i,i'}$$

The differences between the upper and lower bound constraints of $skew_{i,k}$ and $skew_{i',k}$ are exactly the same: $d_{k,i} + d_{i,k}$.

The construction of the merging region requires the computation of two Manhattan arcs as the boundary segments of the merging region. Let $L = d(ss_a, ss_b)$, t_i^a be the sink delay of s_i in T_a, and t_k^b be the sink delay of s_k in T_b. Note that for any sink $s_{i'} \in T_a$, $t_{i'}^a = t_i^a - skew_{i,i'}$. Then, assuming that $0 \leq x \leq 1$, the construction of mr_v on the basis of the delays of $s_{i'}$ and s_k, as well as the skew constraints of $skew_{i',k}$, is equivalent to finding a suitable range of x such that

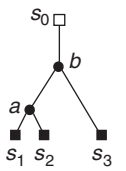
$$\begin{aligned} -d_{i,k} - skew_{i,i'} \leq t_i^a - skew_{i,i'} + r \cdot x \cdot L \cdot \left(C_{T_a} + \frac{c \cdot x \cdot L}{2} \right) - \\ \left(t_k^b + r \cdot (1-x) \cdot L \cdot \left(C_{T_b} + \frac{c \cdot (1-x) \cdot L}{2} \right) \right) \\ \leq d_{k,i} - skew_{i,i'}. \end{aligned}$$

Eliminating $skew_{i,i'}$ from the preceding expression, we obtain

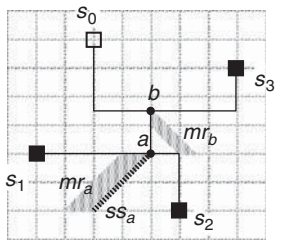
$$\begin{aligned} -d_{i,k} \leq t_i^a + r \cdot x \cdot L \cdot \left(C_{T_a} + \frac{c \cdot x \cdot L}{2} \right) - \\ \left(t_k^b + r \cdot (1-x) \cdot L \cdot \left(C_{T_b} + \frac{c \cdot (1-x) \cdot L}{2} \right) \right) \leq d_{k,i} \end{aligned}$$

the inequalities necessary for the construction of merging region mr_v from s_i and s_k, the respective representative clock sinks of T_a and T_b. Consequently, merging regions constructed on the basis of different pairs of clock sinks from T_a and T_b coincide exactly.

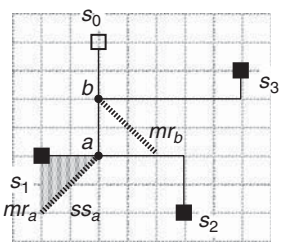
Figure 13.43c shows the tree of merging regions and the useful skew tree that corresponds to the skew commitment $skew_{1,2} = -3$ in Figure 13.21c. First, mr_a (shaded) is constructed on the basis of the merging of s_1 and s_2. The skew from any points in mr_a to s_1 and s_2 falls in the range $[-7, -3] \subset FSR_{1,2} = [-9, -3]$ and is, therefore, feasible. Next, mr_b is constructed from the merging of $ss_a \in mr_a$ and s_3. Note that b is also the singleton child node s_0' of s_0. The skew schedule realized by the UST/DME tree under the path length delay model is $\{t_1 = 6, t_2 = 9, t_3 = 8\}$, which is feasible subject to the constraints given in Figure 13.21. In contrast, we also show a bounded-skew tree with $B = 1$ in Figure 13.43b.



Topology
(a)



BST/DME tree $B = 1$
(b)



UST/DME tree
(c)

FIGURE 13.43

(a) An abstract topology G. (b) The merging tree and bounded-skew tree with $B = 1$ by BST/DME for *G*. (c) The merging tree and useful-skew tree by UST/DME for topology *G* subject to the skew constraints given in Figure 13.21.

For topology generation, we can follow the approach of Greedy-DME. One might be tempted to incorporate the flexibility that Greedy-BST/DME has in allowing merging of subtrees with non-root nodes. However, such a feature will significantly increase the computational complexity. Construction of mr_{e_v}, as defined in the Greedy-BST/DME algorithm, would require the incremental scheduler to uncommit skew commitments made in the previous merging steps. However, that would entail rebuilding the matrix *D*, which has an $O(n^3)$ complexity.

13.3.4 Clock tree optimization

In the preceding section, we focus mainly on the problems of skew scheduling and clock routing, which take into account the timing and wiring aspects of clock network synthesis. As pointed out in Section 13.2, there are other important design aspects of a clock network. This section examines other clock network optimization techniques that address some of these design considerations. Buffer insertion, for example, helps to further improve the timing characteristics of clock signals. Instead of repeating gates (*i.e.,* buffers), clocking control gates can be inserted to turn off flip-flops and their ensuing combinational logic modules when these modules are not required. Besides helping to improve the delay of a clock signal, sizing of buffers and wires can also be very effective in countering skews unintentionally introduced because of variations in manufacturing parameters. Moreover, to further enhance the tolerance of a clock tree to process variations, cross-links can be inserted to provide alternative paths for the clock signal to arrive at selected clock sinks.

13.3.4.1 *Buffer insertion in clock routing*

To drive a large load such as a clock tree, a possible solution is to use cascaded drivers that are of exponentially increasing sizes [Lin 1975]. However, the area requirement and power consumption of such drivers can be prohibitively high. A common solution to this is to break a large clock tree into multiple smaller trees, each driven by a buffer.

As shown in Figure 13.44a, a clock driver/buffer b is modeled as a switch-level RC circuit with a gate capacitance C_b, an output resistance R_b, and an intrinsic delay T_b caused by parasitics capacitance ($= T_b/R_b$) associated with the transistors. A buffer inserted in a long interconnect shields the downstream capacitance C_{down} after the output of the buffer from the upstream resistance R_{up} before the gate input of the buffer as shown in Figure 13.44b. However, it presents a load of C_b, in addition to the capacitance of the upstream interconnect, denoted C_{up}, to the upstream resistance. The new Elmore delay is

$$R_{up}(C_{up} + C_b) + T_b + (R_b + R_{down})C_{down}$$

If $R_{up}C_b + T_b + R_bC_{down} < R_{up}C_{down}$, the insertion of the buffer reduces the overall delay of the long interconnect.

Clearly, buffer insertion can play an important role in minimizing the clock phase delay, which is defined as the maximum among the sink delays. Even when delay minimization is not the main goal of clock tree synthesis, buffer insertion may help in reducing the overall wiring cost. In zero-skew routing, for example, it is always desirable to merge two subtrees with similar sink delays; when the sink delays differ greatly, wire snaking (see Figure 13.25) is commonly used to balance them by making a faster clock signal path slower. With buffer insertion, the slower clock signal path can be made faster, thereby eliminating the need for wire snaking. Moreover, compared with a clock tree

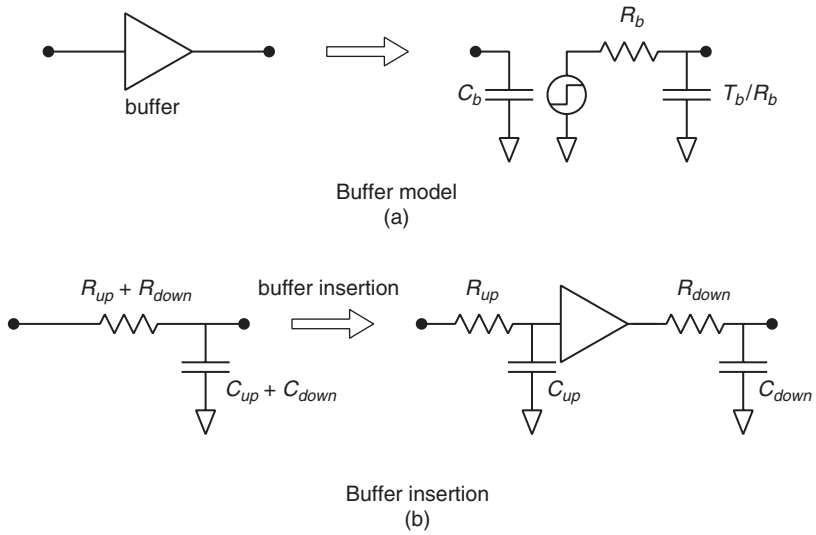


FIGURE 13.44

(a) A switch-level model for a clock buffer/driver. (b) The insertion of a buffer to break up a long interconnect.

with drivers only at the root node, it is easier to satisfy the rise/fall time constraints with a buffered clock tree.

With buffer insertion, clock power can also be reduced because of the reduced load presented to the clock driver and intermediate clock buffers. It is no longer necessary to have huge clock drivers, which add to the power consumption. The reduction of capacitive load also reduces the current flowing through the clock driver, improving the reliability of the interconnects near the driver output. Moreover, the current demand is now more evenly distributed across the entire chip, potentially reducing the current-induced noise in the power supply network.

The buffered clock tree construction algorithm in [Vittal 1995] is an extension of the Greedy-DME algorithm to consider the insertion of intermediate clock buffers during the construction of a zero-skew routing tree. In each merging step, three sets of possible locations for embedding an internal node of the abstract topology are computed. One of the three sets is the merging segment as in the case of DME. The other two sets correspond to the possible locations of the internal node when a buffer is inserted to drive one of the child subtrees.

Consider two subtrees T_a and T_b rooted at a and b, respectively, as shown in Figure 13.45. The merging segment ms_v of v, the parent node of a and b, can be computed as in the DME algorithm, and it corresponds to the feasible locations of v when no buffer is inserted.

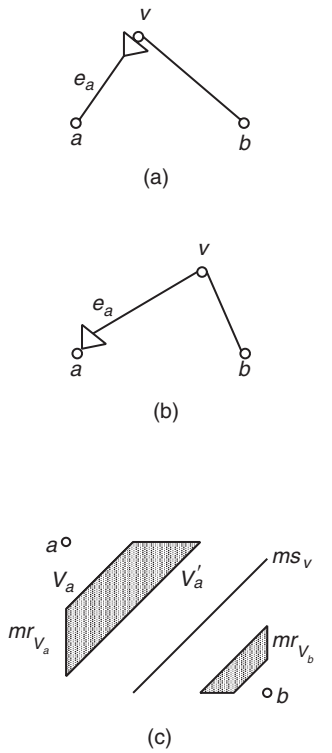


FIGURE 13.45

Insertion of a buffer at different locations along the edge e_a to drive T_a alone.

A buffer may be inserted at v to drive only T_a, but not T_b, through e_a, as shown in Figure 13.45a. The Manhattan arc V_a, as shown in Figure 13.45c contains the set of feasible locations of v when zero-skew is achieved for such a buffer location. Alternatively, the buffer may be inserted at a as shown in Figure 13.45b, and the Manhattan arc V'_a, as shown in Figure 13.45c, corresponds to the set of candidate locations of v for this alternate buffer location. In this example, the insertion of a buffer at v results in a longer sink delay than inserting the buffer at a. Consequently, a is closer to V_a than to V'_a.

Because V_a and V'_a represent the locations of v when a buffer is inserted at the extreme possible locations (at the start and end of edge e_a, respectively), the "merging" region mr_{v_a} bounded by V_a and V'_a corresponds to all possible locations of v when the buffer is inserted at any point along e_a. It is important to note that the region defined by a and V_a in Figure 13.45c contains the possible locations for buffer insertion along e_a. Also note that $|e_a|$ depends on the buffer position.

Similarly, a buffer may be inserted to drive T_b alone. The merging region mr_{v_b} in Figure 13.45c captures the set of feasible locations to embed v when

a buffer is inserted to drive T_b. Although not shown here, it is possible that mr_{v_a}, mr_{v_b}, and ms_v may overlap and that their relative positions may vary. Moreover, we can construct merging segments/regions on the basis of other buffering combinations; we may, for example, insert buffers into a and b simultaneously.

In all of these merging operations, it is important to distinguish between the total capacitance of a subtree and the capacitance of a DC-connected subtree. Consider the buffered interconnect in Figure 13.44b. There are two DC-connected subtrees, with one being the upstream of the buffer and the other being the downstream of the buffer. The total capacitance of the buffered interconnect is $C_{up} + C_{down} + C_b$,[2] whereas the capacitance of the DC-connected subtree corresponding to the upstream of the buffer is simply $C_{up} + C_b$. In many of the equations for computing merging segments/regions in Section 13.3.3, C_{T_a} and C_{T_b} are, in fact, the capacitances of the DC-connected subtrees rooted at a and b, respectively. For quantifying the wiring cost of the entire clock tree, we, of course, use the total capacitance of the tree. For clarity, we will use $C_{T_a}^{tot}$ to denote the total capacitance of the tree rooted at a.

The buffered clock tree construction algorithm follows the flow of the Greedy-DME algorithm with the following modifications. Instead of the use of only wire length to define merging cost, the cost of merging is a weighted combination of multiple factors such as total wire length, total buffer size, and rise time. As each internal node has multiple sets of feasible locations for its embedding (as in the case of BST/DME and UST/DME), the sink delays are not determined when these feasible locations are computed. The sink delays are determined at the next level of the merging step when the respective merging segments or merging regions (or sampling segments for ease of implementation) of the two sibling nodes that yields locally minimum zero skew merging cost are selected. Besides considering buffering in a merging operation, a buffer can also be inserted to drive the merged subtree if the sinks of the subtree do not have sharp clock edges (*i.e.,* long rise/fall time).

Most buffered clock tree construction algorithms place an upper-bound constraint on the difference in the numbers of buffers in any two source-to-sink paths in a clock tree. This is a preventive measure to minimize the effects of variations in the electrical parameters of clock buffers on the clock delays and, hence, the clock skew of the two paths. In the most restrictive case, all source-to-sink paths go through the same number of buffers. Moreover, buffers inserted at the same level are of the same size. Although these restrictions may affect the optimality in terms of signal delay and total wire length, they greatly reduce the sensitivity to process variations of clock skew.

One example of such algorithms is that in [Chen 1996], where instead of considering buffer insertion at each merging step as in [Vittal 1995], buffers

[2]For simplicity, we have ignored the parasitic capacitance intrinsic to the buffer.

are inserted at the roots of all subtrees simultaneously. Recall that in DME, we maintain F, a forest of subtrees. Starting with a forest of singleton subtrees, several iterations of DME-based zero-skew merging are performed until $|F|$ has been reduced by 2^k for some k, which is determined on the basis of the clock buffer strength. The stopping criterion could also be based on the rise/fall time constraint or the maximum capacitance of the subtrees in F. At this point, buffers are inserted at the roots of all subtrees in F. This is akin to clustering of nodes, followed by buffer insertion to drive each cluster.

An inserted buffer may not be connected to the root node directly. Instead, a wire may be used to connect from the buffer output to the root of the subtree such that all subtrees in S have equal sink delay. In other words, the root node of a subtree may have a tilted rectangular region to represent the set of possible locations for the inserted buffer. A future merging step that involves such a subtree will then be based on a tilted rectangular region instead of a merging segment.

Remark: Post-Silicon Tunable Buffers: Skews induced by process variations can result in significant yield loss. To address this problem, circuit designers have deployed so-called ***post-silicon tunable*** (PST) circuitry in modern commercial processor chips such as Intel P4 and Itanium [Geannopoulos 1998; Tam 2000]. The concept of post-silicon tuning is illustrated in Figure 13.46, where some of the clock buffers are delay-tunable. The delay of any of these buffers can be adjusted by activating an appropriate number of capacitors between the two inverters that form the buffer (see the schematic of a delay-tunable buffer in Figure 13.46). The post-silicon adjustment can be performed dynamically with on-chip de-skew

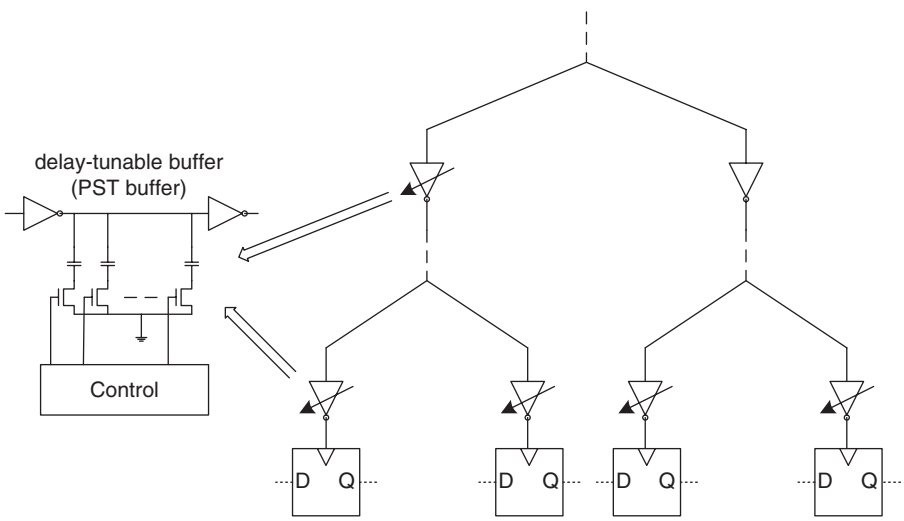


FIGURE 13.46

A clock tree with post-silicon tuning circuitry.

circuitry, whereas a fuse-based solution allows only one-time adjustment. Other post-silicon tunable techniques involve voltage biasing of the buffers.

13.3.4.2 *Clock gating*

So far, we have considered designs in which all clocked modules always receive a clock signal. However, for modules that are idle, it is not necessary to send them a clock signal. Suppose we isolate the modules from the clock source when they are in the idle mode, then the power consumption caused by the capacitance of the clock pins in these modules can be reduced. Moreover, because the flip-flops in these modules are not triggered, the output nodes of these flip-flops retain their values throughout the idle period. Consequently, the combinational logic following these flip-flops does not have any switching activity during the idle period. In other words, there is no active power consumed by idle modules. This is the clock-gating technique presented in Section 13.2.4.

In clock gating, clocking control gates (or clock gates) are inserted along with buffers in a clock tree. Consider for example a clock tree as shown in Figure 13.47, where internal node v is a candidate buffer/gate location. The activity pattern associated with each node over 6 time units is also shown, with a dark square indicating that the node should be active in this time unit and an empty square indicating an idle time unit. Note that a time unit could be multiple clock cycles. The activity pattern of an internal node is derived from the activity patterns associated with its child nodes. With a "1" to indicate an active time unit and a "0" to indicate an idle time unit, the bitwise-OR of the activity patterns of two child nodes generates the activity pattern of the parent node. Node v, for example, is active when at least one of its child nodes (*i.e.*, a and b) is active, as shown by the respective activity patterns in Figure 13.47.

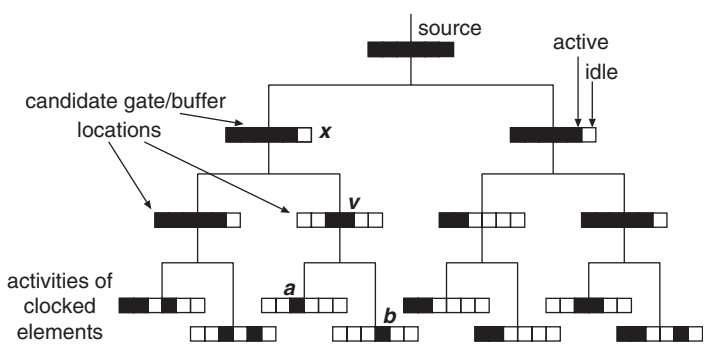


FIGURE 13.47

A clock tree topology with activity patterns for modules and candidate buffer/gate locations.

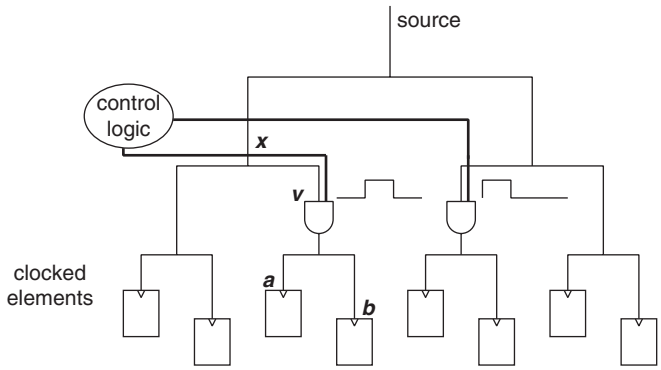


FIGURE 13.48

A gated clock tree, with two inserted clocking control gates.

The gated clock tree as shown in Figure 13.48 is the result of inserting two clock gates in the clock tree given in Figure 13.47. The clock gate at v must be active and allow the clock signal to propagate through when at least one of its sinks (*i.e.*, a or b) is active. Therefore, a gating-signal control logic module that generates control signals to turn on or off the clock gates is required.

Although clock gating reduces dynamic power, it has an overhead of the gating-signal control logic. Moreover, as long as the clock signal arrives at a clock gate, the clock gate contributes to power consumption because of its gate capacitance or the switching of its internal nodes. The other downside of clock gating is the clock skew caused by a clock gate. These are the main issues handled by gated-clock synthesis algorithms in [Téllez 1995; Oh 2001; Chen 2002; Chao 2008].

To construct a zero-skew gated-clock tree, let us again assume that we are given an abstract topology. The tasks here are to embed the internal nodes and to determine where buffers and clock gates should be inserted. We will now describe an adaptation and simplification of the approaches described in [Téllez 1995; Oh 2001; Chen 2002; Chao 2008]. The approach also closely resembles that for the construction of a buffered clock tree. Again, let v be the parent node of nodes a and b. When we merge subtrees rooted at a and b, we allow the following three choices at the root of e_a:

1. No buffer or clock gate is inserted at the root of e_a.
2. A buffer is inserted at the root of e_a.
3. A clock gate is inserted at the root of e_a. Here, we assume that similar to a buffer, a clock gate is also modeled with a switch-level RC model with gate capacitance C_{cg}, output resistance R_{cg}, and intrinsic delay T_{cg}.

We similarly afford the three choices at the root of e_b. There are, therefore, altogether 9 different ways of merging a and b, resulting in 9 zero-skew merging segments. As in all other methods that generate multiple merging segments/sampling segments when merging two subtrees, it is prudent to keep only a

fixed number of sampling segments at each internal node. It is typical that the subset of sampling segments retained at each internal node is selected on the basis of the merging cost, which has to be modified significantly to account for dynamic power reduction.

Without clock gating, the power consumption of a clock tree can be obtained directly from the wiring cost (and buffering cost if buffer insertion is considered). With the insertion of a clock gate, the switching activity of the subtree after the clock gate is greatly reduced. Consider node v in Figure 13.48, the capacitance $C_{T_v}^{tot}$ *after* the clock gate switches only a third of the time. In a non-gated-clock tree, the power consumption of the subtree rooted at v has a dynamic power of $C_{T_v}^{tot} V_{DD}^2 f_{clk}$, whereas the gated subtree in Figure 13.48 has a dynamic power of only $C_{T_v}^{tot} V_{DD}^2 f_{clk}/3$. We call $C_{T_v}^{tot}/3$ the effective switching capacitance of the subtree rooted at v, with the coefficient of a third being the activity factor, denoted α_v.

In the synthesis of a gated-clock tree, the minimization of the effective switching capacitance of the entire clock tree is the main objective. To compute the effective switching capacitance of a subtree, we decompose the total capacitance of the subtree into two parts, gated capacitance and ungated capacitance. Take node x in Figure 13.48 for example—its gated capacitance, denoted GC_{T_x}, is $C_{T_v}^{tot}$. The ungated capacitance of x, denoted UGC_{T_x}, is $C_{T_x}^{tot} - GC_{T_x} = C_{T_x}^{tot} - C_{T_v}^{tot}$. Let the effective switching capacitance of the gated capacitance be $GC_{T_x}^{eff}$, which in this example is $C_{T_v}^{tot}/3$. Then, the effective switching capacitance at x, denoted $C_{T_x}^{eff}$, is $UGC_{T_x} + GC_{T_x}^{eff}$.

If we insert a buffer at x, the ungated capacitance, as well as the effective switching capacitance, at x are increased by the gate capacitance of the buffer C_b. If instead of a buffer, we insert a clock gate at x, the gated capacitance would be increased by UGC_{T_x}, whereas the ungated capacitance at x reduces to simply C_{cg}, the gate capacitance of the clock gate. Moreover, the effective switching capacitance at x is reduced by $(1 - \alpha_x) UGC_{T_x} - C_{cg}$. The following ordered operations update these capacitances correctly when a clock gate is inserted at x:

1. $C_{T_x}^{tot} \leftarrow C_{T_x}^{tot} + C_{cg}$;
2. $GC_{T_x} \leftarrow GC_{T_x} + UGC_{T_x}$;
3. $GC_{T_x}^{eff} \leftarrow GC_{T_x}^{eff} + \alpha_x UGC_{T_x}$;
4. $UGC_{T_x} \leftarrow C_{cg}$;
5. $C_{T_x}^{eff} \leftarrow UGC_{T_x} + GC_{T_x}^{eff}$.

Note that the insertion of a clock gate or a buffer creates a new DC-connected subtree, with C_{cg} or C_b, respectively, being the new C_{T_x}, which would be required for the computation of merging segments at the parent node.

The merging cost is typically defined to be some combination of the wire length $|e_a| + |e_b|$, effective switching capacitance, and possibly other metrics. An important metric that we have left out in the preceding discussion is the switching activity (and effective switching capacitance) of various clock gate control signals. Consequently, it is unwise to insert a clock gate right before a register that is enabled most of the time. To incorporate the power consumption of

gating signals, we may assume a centralized gating-signal control logic and use its distance from a clock gate to estimate the wire capacitance of a gating signal.

As in the case of Greedy-DME, we may interleave the generation of abstract topology with the computation of merging segments/sampling segments. However, the efficacy of such an approach relies on the existence of neighboring subtrees that are of similar switching activity. Therefore, it is important that the high-level synthesis, logic-level synthesis, and physical-level synthesis steps all work hand-in-hand with gated-clock tree construction. All in all, the fundamental difficulty lies in the prediction of the activity pattern of various modules. Although the data activity can be quite well characterized for DSP circuits and microprocessors, it is more difficult to generate activity patterns for a general circuit/system.

13.3.4.3 *Wire sizing for clock nets*

In this section, we deal with the problem of assigning appropriate widths to wires in a clock tree to minimize the clock skew, the clock delay, and the sensitivity of the clock tree to process variations. The constraint on the maximum width of a wire is typically imposed by the available routing resources, whereas the constraint on the minimum wire width depends on the fabrication technology. Moreover, the maximum allowable current density through a wire also provides a lower bound for the wire width, so that the wire can withstand the wear-out phenomenon caused by electromigration. Note that a long wire may be divided into several segments, and each segment may have different upper and lower bounds.

Remark: Mitigating Electromigration: Wire sizing to counter electromigration can be easily handled by DME-based clock routing algorithms. When nodes a and b are merged at the parent node v, the total capacitances of subtrees T_a and T_b, which provide respective estimates of the amounts of current flow in e_a and e_b, can be used to determine for e_a and e_b appropriate wire widths that are tolerable to electromigration. Zero-skew, bounded-skew, or useful-skew merging can then be carried out with appropriate wire widths for e_a and e_b.

13.3.4.3.1 Delay sensitivity and delay minimization

From Section 13.3.2, the Elmore delay from the clock driver at the source s_0 to sink s_i in an RC tree is

$$t_i = R_d \cdot C_{T_{s_0}} + \sum_{e_v \in Path(s_0, s_i)} |e_v| \cdot \frac{r}{w_{e_v}} \cdot \left(\frac{|e_v| \cdot w_{e_v} \cdot c_a}{2} + C_{T_v} \right)$$

where R_d is the output resistance of the clock driver. For simplicity, this formulation ignores the fringing capacitance of wires, which can be added easily. Taking the partial differential $\frac{\partial t_i}{\partial w_{e_v}}$ for any edge e_v along the s_0–s_i path,

$$\frac{\partial t_i}{\partial w_{e_v}} = R_d \cdot c_a \cdot |e_v| - \frac{|e_v| \cdot r \cdot C_{T_v}}{w_{e_v}^2} + \sum_{e_u \in Ans(e_v)} \frac{|e_u| \cdot r \cdot c_a \cdot |e_v|}{w_{e_u}} \tag{13.10}$$

where $Ans(e_v)$ contains the path from s_0 to the parent node of v. If e_v is not along $Path(s_0, s_i)$,

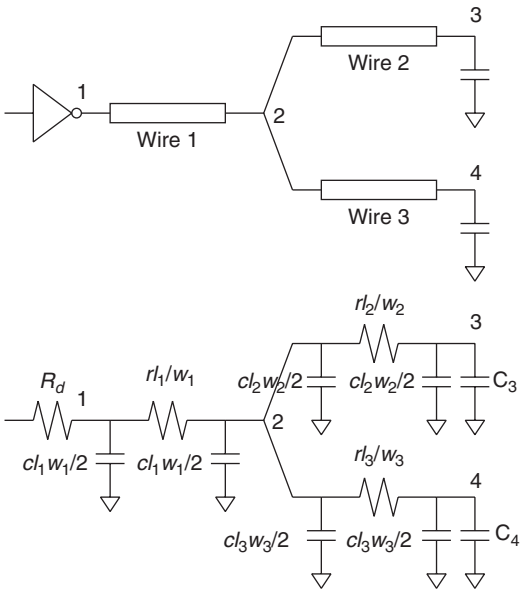


FIGURE 13.49

A clock tree with two sinks.

$$\frac{\partial t_i}{\partial w_{e_v}} = R_d \cdot c_a \cdot |e_v| + \sum_{e_u \in Ans(e_v) \cap path(s_0, s_i)} \frac{|e_u| \cdot r \cdot c_a \cdot |e_v|}{w_{e_u}} \tag{13.11}$$

Let us illustrate the computation of partial derivatives with a simple example as shown in Figure 13.49. The RC tree has three segments with wire widths of w_1, w_2, and w_3, respectively. Every wire segment of length l_i is modeled as π-type RC circuit with a resistance of rl_i/w_i and a total capacitance of cl_iw_i, where r is the resistance per square and c is the capacitance per unit area. The partial derivatives of t_3 and t_4 with respect to w_2 are:

$$\frac{\partial t_3}{\partial w_2} = R_d \cdot c \cdot l_2 - \frac{l_2 \cdot r \cdot C_3}{w_2^2} + \frac{l_1 \cdot r \cdot c \cdot l_2}{w_1}$$

$$\frac{\partial t_4}{\partial w_2} = R_d \cdot c \cdot l_2 + \frac{l_1 \cdot r \cdot c \cdot l_2}{w_1}$$

The partial differential $\frac{\partial t_i}{\partial w_{e_v}}$ evaluated at W, the currently assigned wire widths, captures the *delay sensitivity* of t_i with respect to a change in w_{e_v}. A positive sensitivity indicates that the delay increases if we widen e_v, whereas a negative sensitivity indicates that the delay decreases. When all sinks have zero delay sensitivity, the clock net is extremely tolerable to process variations, because the sink delays are not sensitive to these changes.

Having zero delay sensitivity means that we minimize all the sink delays. A sink delay can be locally minimized by setting $\frac{\partial t_i}{\partial w_{e_v}} = 0$ and solving for w_{e_v} while keeping the widths of other wires intact. Consequently, wires closer to the root should have wider wire width, because they see a larger downstream capacitance. The term $R_d \cdot c_a \cdot |e_v|$ in the partial derivative keeps the wire width from getting too large. It is also a common practice to impose an upper-bound constraint on the wire width.

It should also be obvious that the larger the downstream capacitance seen by a wire, the larger the delay sensitivity. Consequently, buffer insertion, which decouples downstream capacitance from the upstream resistance, can also greatly desensitize a clock net. Similarly, we can also define the partial derivatives of a sink delay with respect to buffer sizes and perform appropriate sizing of buffer/driver to reduce sink delay. Although the focus here is wire sizing, similar techniques can be used to perform buffer sizing. In fact, many techniques that are based on delay sensitivity perform buffer and wire sizing together [Wang 2005; Guthaus 2006].

Similar to wire sizing for mitigating electromigration, wire sizing for delay minimization can be easily incorporated in a DME-based clock routing algorithm, as in [Edahiro 1993b]. Instead of operating on an abstract topology, the heuristic approach in [Edahiro 1993b] operates on a zero-skew routing tree, and the reason for that will soon be apparent.

Consider the merging of two zero-skew subtrees T_a and T_b at the parent node v, with w_{e_a} and w_{e_b} being the wire widths of e_a and e_b, respectively. Let $L = d(ms_a, ms_b)$ be the distance between the merging segments of a and b and $x \cdot L$ be the distance of the merging segment of v from a, where $0 \leq x \leq 1$, assuming no detour wiring. Then, x can be computed with the following expression:

$$x = \frac{t_b - t_a + r \cdot L \cdot (C_{T_b}/w_{e_b} + c \cdot L/2)}{r \cdot L \cdot (c \cdot L + C_{T_a}/w_{e_a} + C_{T_b}/w_{e_b})}$$

Taking the derivative of x with respect to w_{e_a} yields

$$\frac{\partial x}{\partial w_{e_a}} = \frac{C_{T_a}/w_{e_a}}{c \cdot L + C_{T_a}/w_{e_a} + C_{T_b}/w_{e_b}} \left(\frac{x}{w_{e_a}}\right)$$

Recall that an optimal width assignment actually depends on both upstream resistance and downstream capacitance. Although downstream capacitances are known in the bottom-up process, we have to approximate the resistance in the upstream. We use the given zero-skew routing tree to approximate the upstream resistance from the clock source s_0 to v in the final routing tree by assuming, for example, nominal wire widths for the upstream edges. Let R_{up} denote the approximate upstream resistance. Then, the estimated delay from s_0 to a sink in T_a is

$$R_{up} \cdot c \cdot L \cdot (w_{e_a} \cdot x + w_{e_b} \cdot (1 - x)) + r \cdot x \cdot L \cdot \left(\frac{C_{T_a}}{w_{e_a}} + \frac{c \cdot x \cdot L}{2}\right) + K$$

where K is a constant independent of x and w_{e_a}. Taking the derivative of the delay with respect to w_{e_a} and setting the derivative to zero, we obtain

$$R_{up} \cdot c \cdot w_{e_a}^2 \left(\frac{w_{e_b}}{w_{e_a}} C_{T_a} \left(2 - \frac{w_{e_b}}{w_{e_a}} \right) + C_{T_b} + w_{e_b} \cdot c \cdot L \right) = r \cdot C_{T_a} \cdot C_{T_b} + r \cdot c \cdot w_{e_b} \cdot (1 - x) \cdot L \cdot C_{T_a}$$

Assuming that wire capacitances $w_{e_b} \cdot c \cdot L$ and $w_{e_b} \cdot c \cdot (1 - x) \cdot L$ are much smaller than C_{T_b}, we discard them and obtain the following expression:

$$R_{up} \cdot c \cdot w_{e_a}^2 \left(\frac{w_{e_b}}{w_{e_a}} C_{T_a} \left(2 - \frac{w_{e_b}}{w_{e_a}} \right) + C_{T_b} \right) = r \cdot C_{T_a} \cdot C_{T_b}$$

A similar expression can be obtained by examining how the delay of a sink in T_b is affected by e_b:

$$R_{up} \cdot c \cdot w_{e_b}^2 \left(\frac{w_{e_a}}{w_{e_b}} C_{T_b} \left(2 - \frac{w_{e_a}}{w_{e_b}} \right) + C_{T_a} \right) = r \cdot C_{T_a} \cdot C_{T_b}$$

Equating the two preceding equations yields the following analytical formula for the wire width of e_a and e_b:

$$w_{e_a} = w_{e_b} = \min \left\{ \max \left\{ w_{\min}, \sqrt{\frac{r \cdot C_{T_a} \cdot C_{T_b}}{R_{up} \cdot c \cdot (C_{T_a} + C_{T_b})}} \right\}, w_{\max} \right\}$$

where $w_{\min}$ is the minimum wire width allowed by the technology or specified by electromigration constraints, and $w_{\max}$ is typically determined by the available wiring resources.

Because the newly computed wire lengths and widths may differ greatly from the original clock routing tree, it is recommended that the process be repeated for a few iterations. Because the wire widths are determined on the basis of delay sensitivity, the sensitivity of the clock tree to process variations is reduced indirectly after the iterative procedure. However, be aware that the iterations may not converge. In other words, wire lengths and widths may keep changing in successive applications of the modified DME algorithm. Restricting the procedure to just a few iterations provides a good compromise between solution quality and runtime efficiency.

Although the bottom-up wire sizing approach described in the preceding paragraphs assumes continuous wire width, it can also be modified to consider discrete wire widths. Because it is not possible to achieve arbitrary precision during fabrication, it is better to have the algorithm explicitly synthesize a clock tree with discrete wire widths to eliminate undesirable skew caused by the post-synthesis mapping of continuous widths to discrete widths.

Remark: Skew Sensitivity: What is truly of interest is *skew sensitivity*, which measures how a change in wire width can affect the clock skew. In particular, skew sensitivity caused by process variations can be used to measure how

reliable a clock tree is. Skew sensitivity is simply the difference of delay sensitivity. Take the RC tree in Figure 13.49 for example, the skew sensitivity of $skew_{3,4} = t_3 - t_4$ is

$$\frac{\partial}{\partial x_2}(t_3 - t_4) = -l_2 \cdot r \cdot C_3 / w_2^2 \tag{13.12}$$

It should be obvious for any two sinks, only the subtree rooted at the youngest common ancestor of the two sinks contributes to the skew sensitivity. Because the two sinks share the same path from the clock source to the youngest common ancestor, any changes along the path result in the same changes in the two sink delays, which negate each other in the computation of skew.

However, because of the definition of global clock skew as $gskew = \max_{i,j}|t_i - t_j|$, it is very costly to compute global skew sensitivity exactly. The exact approach would entail the computation of the changes in the worst-case global clock skew for each wire under consideration. Typically, an approximation method such as that in [Xi 1995] would be used.

13.3.4.3.2 Wire sizing with dynamic programming

No treatment of wire sizing is complete without the inclusion of a very popular technique to solve difficult programs in EDA, namely, dynamic programming. This solution method was applied to buffer insertion in 1990 by van Ginneken [van Ginneken 1990] for delay minimization of signal nets and has since been generalized to perform wire sizing (and buffer sizing) as well [Lillis 1995]. Although the solution we will see here is not in the same mold as those in [van Ginneken 1990; Lillis 1995], it is still based on the fundamental principle of building and enumerating solutions of larger problems from solutions of smaller problems.

Here, we assume that we are sizing an unbuffered zero-skew clock tree T. Let $\gamma = (C, t)$ denote a solution at a node v in T, where C is the total capacitance rooted at v and t is the delay from v to a sink in T_v. Consider a clock sink s_i. There is only one solution $\gamma_{s_i} = (C = c_{s_i}^g, t = 0)$ associated with it, where $c_{s_i}^g$ is the gate capacitance of s_i.

Now, consider the edge e_{s_i} connecting s_i to its parent node a. We refer to the tree with e_{s_i} as the root edge as $T_{e_{s_i}}$. Assume that a wire width of w is assigned to it. Then, $\gamma_{e_{s_i}}(w) = (C^+, t^+)$, the solution that corresponds to such a wire width assignment has the following capacitance and sink delay:

$$C^+ = C + |e_{s_i}| \cdot w \cdot c \quad \text{and} \quad t^+ = t + \frac{r \cdot c \cdot |e_{s_i}|^2}{2} + \frac{r \cdot |e_{s_i}|}{w} \cdot C$$

where $(C, t) = \gamma_{s_i}$. Because we can pick any width in the range $w_{\min} \leq w \leq w_{\max}$ to form such a solution, the solution space associated with $T_{e_{s_i}}$ can be captured by treating the capacitance and sink delay in $\gamma_{e_{s_i}}(w)$ as the coordinates of a point on a 2-D plane, of which the x-axis is capacitance and the y-axis

is sink delay. The solution spaces associated with $T_{e_{s_i}}$ and $T_{e_{s_j}}$, which are connected at a, are similar to the leftmost plot in Figure 13.50.

Let $\Gamma_{e_{s_i}}$ and $\Gamma_{e_{s_j}}$ denote the solution spaces associated with $T_{e_{s_i}}$ and $T_{e_{s_j}}$, respectively. The merging at a is feasible if, and only if, there exist $\gamma_{e_{s_i}}(w) = (C_i, t_i) \in \Gamma_{e_{s_i}}$ and $\gamma_{e_{s_i}}(w') = (C_j, t_j) \in \Gamma_{e_{s_i}}$ with $t_i = t_j$. The solution associated with a obtained from the merging of $\gamma_{e_{s_i}}(w)$ and $\gamma_{e_{s_i}}(w')$ is $\gamma_a(w, w') = (C_i + C_j, t_i = t_j)$. The solution space associated with a, denoted Γ_a, is again similar to that shown in the leftmost plot of Figure 13.50. Note that $\gamma_{e_{s_i}}(w_{\min})$ has the largest sink delay and $\gamma_{e_{s_i}}(w_{\max})$ the smallest. Let $t(\gamma)$ denote the sink delay in solution γ. Then, the sink delay in Γ_a must lie in the range $[t(\gamma_{e_{s_i}}(w_{\max})), t(\gamma_{e_{s_i}}(w_{\min}))] \cap [t(\gamma_{e_{s_j}}(w_{\max})), t(\gamma_{e_{s_j}}(w_{\min}))]$.

Now, if we pick any $\gamma = (C, t) \in \Gamma_a$, we can construct a new (partial) solution space associated with T_{e_a} based on γ by assigning wire width $w_{\min} \leq w \leq w_{\max}$ to e_a. The construction of such a solution, denoted as $\Gamma_{e_a}(\gamma)$, is similar to that for $T_{e_{s_i}}$ from γ_{s_i}. Consequently, we would again obtain a solution space that is similar to the leftmost plot in Figure 13.50. However, to obtain the overall solution space associated with T_{e_a}, we would have to consider all solutions in Γ_a. In other words, $\Gamma_{e_a} = \cup_{\gamma \in \Gamma_a} \Gamma_{e_a}(\gamma)$, which is typically a 2-D region, as shown in the middle plot of Figure 13.50. It is important that we keep track of which solutions from the two edges contribute to a solution in the parent node. The arrows from the leftmost plot to the middle plot in Figure 13.50 depict this relationship.

Clearly, the computation of the wire-sizing solution spaces of internal nodes of the zero-skew routing tree quickly becomes quite unwieldy. To overcome this problem, a sampling approach [Tsai 2003] can be used. Instead of computing a continuous 2-D solution space, a sample solution that is represented by a set of horizontal line segments is calculated. Each horizontal line segment corresponds to a range of capacitances $C_{\min} \leq C \leq C_{\max}$ that achieve a particular sink delay t. There may be a few sampling segments for a particular sink delay t, because there are a few ranges of capacitances that could result in the same delay. We show two sampling segments in the middle plot of Figure 13.50.

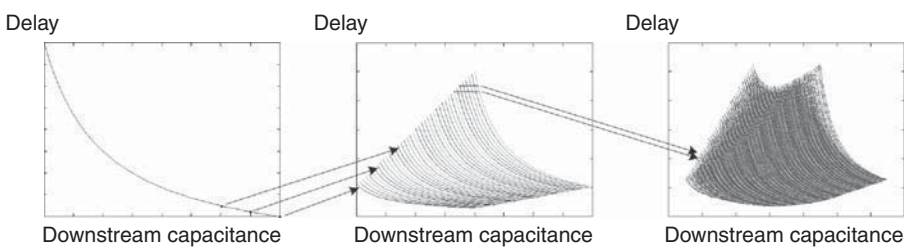


FIGURE 13.50

Wire sizing solution spaces for various nodes in a zero-skew routing tree.

The sampling of Γ_a is quite straightforward. We will now illustrate how the sampled Γ_{e_a} can be computed. It is best that we compute the sampled Γ_{e_a} together with sampled Γ_{e_b}, where a and b are siblings in the zero-skew routing tree. Doing so allows the sink delays to be sampled within the same range and facilitates the merging of the two sampled solution spaces to achieve zero-skew. In other words, the first step is to determine the minimum and maximum sink delays in Γ_{e_a} and Γ_{e_b} given the sampled solution spaces Γ_a and Γ_b. This is trivial, because for each $\gamma = (C, t) \in \Gamma_a$, the minimum (maximum) sink delay of T_{e_a} can be determined by assigning the maximum (minimum) wire width to e_a. The minimum (maximum) sink delay in sampled Γ_{e_a} is, therefore, the smallest (largest) among the minimum (maximum) sink delays obtained in this fashion.

Let $t_{e_a}^{\max}$ and $t_{e_b}^{\max}$ ($t_{e_a}^{\min}$ and $t_{e_b}^{\min}$) denote the maximum (minimum) sink delays in Γ_{e_a} and Γ_{e_b}, respectively. The sampled delays in Γ_{e_a} and Γ_{e_b} would, therefore, lie in the range $[t_{e_a}^{\min}, t_{e_a}^{\max}] \cap [t_{e_b}^{\min}, t_{e_b}^{\max}]$. Considering a sampled delay t^+ that is in this range, we now have to compute the range of capacitances that can achieve this delay. For each (C, t) from the sampled Γ_a, we can solve for w, the wire width of e_a, that achieves the sampled delay t^+ as follows:

$$w = \frac{r \cdot |e_a| \cdot C}{t^+ - t - \frac{r \cdot c \cdot |e_a|^2}{2}}$$

If $w_{\min} \leq w \leq w_{\max}$, $C^+ = C + |e_a| \cdot w \cdot c$ is in the capacitance range. The capacitance range(s) for the sampled delay t^+ is obtained by clustering the C^+'s obtained in this fashion into appropriate group(s). The solution space Γ_{e_a} is similar to that shown in the rightmost plot of Figure 13.50.

The sampled Γ_{e_b} can be obtained in a similar fashion, and together, Γ_{e_a} and Γ_{e_b} can be merged to obtain Γ_v, where v is the parent node of a and b. It is again important for each solution in Γ_v to keep track of the contributing solutions from Γ_{e_a} and Γ_{e_b}. The process continues until Γ_{s_0} is obtained.

Among the sampled delays in Γ_{s_0}, the one that meets the target delay and has the lowest capacitance is chosen. Because we have kept track of the solutions from the child nodes that contribute to a solution of the parent node, we can easily perform a top-down traversal to determine the wire width of each edge in the zero-skew routing tree.

Remark: Dynamic Programming for Delay Minimization: The biggest difference between wire sizing for delay minimization and zero-skew is that for delay minimization, the dynamic programming approach can apply pruning rules to weed out suboptimal solutions, greatly reducing the search space. Consider two solutions (C, t) and (C', t') of a tree, where the delays t and t' correspond to the maximum sink delays in these two solutions. If $C < C'$ and $t < t'$), there is no reason to consider (C', t') any further in the bottom-up process, because solutions that are built on top of it are always inferior to corresponding solutions constructed upon (C, t).

13.3.4.4 *Cross-link insertion*

Clock trees provide only one unique path from a clock source to any clock sink. Because there is a lack of redundancy in a tree structure, it is difficult to compensate for the effects of variations. The cross-link insertion technique is a promising approach that introduces redundancy while retaining the low wiring cost of a clock tree. The insertion of cross-links or interconnect wires between some nodes of a clock tree reduces the skew variability. We show examples of such clock trees in Figures 13.15 and 13.51.

Given an RC network $G = (V, E)$, we can decompose the network into a spanning tree $T = (V, E_T)$ and a set of link edges (chords) $E_L = E \backslash E_T$. In Figure 13.51 for example, the only edge in E_L is the cross-link l inserted between nodes u and w. Let R_l and C_l be the resistance and capacitance of l, it can be represented by a π-type RC element (see Figure 13.18) with two capacitors $C_l/2$ at the two ends of a resistor R_l.

Let t_i denote the Elmore delay at any node i of $T = (V, E_T)$. With the link l inserted between u and w, the delay at node i changes. We examine the changes caused by the additions of the capacitors and resistor separately. The addition of only the capacitors is fairly straightforward, because the topology remains a tree. The Elmore delay at node i after the addition of capacitors at u and w, denoted as $\tilde{t}_i$, is

$$\tilde{t}_i = t_i + \frac{C_l}{2}(R_{i,u} + R_{i,w}),$$

where $R_{i,u} = R_{Path(s_0,i) \cap Path(s_0,u)}$ $(R_{i,w} = R_{Path(s_0,i) \cap Path(s_0,w)})$ is the common resistance between the paths $Path(s_0, i)$ and $Path(s_0, u)$ $(Path(s_0, i)$ and $Path(s_0, w))$.

When the resistor R_l is inserted, the delay at node i is changed to $\hat{t}_i$ from $\tilde{t}_i$ as follows [Chan 1990]:

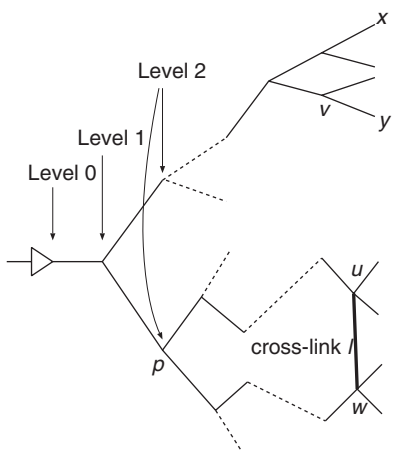


FIGURE 13.51

A clock tree with a cross-link inserted.

$$\hat{t}_i = \tilde{t}_i - \frac{\tilde{t}_u - \tilde{t}_w}{R_l + r_u - r_w} r_i,$$

where resistance r_i is the Elmore delay at node i when the capacitances at nodes u and w are $C_u = 1$ and $C_w = -1$, respectively, and all other node capacitances are zero.

Example 13.4 Adding a Cross-Link: Consider the addition of a cross-link between nodes v and x in Figure 13.17, as shown in Figure 13.52. Let the link capacitance and resistance be C_l and R_l, respectively. First, we add the link capacitance and update the Elmore delays of various nodes:

$$\tilde{t}_y = t_y + \frac{C_l}{2}(R_{y,v} + R_{y,x}), \quad \tilde{t}_z = t_z + \frac{C_l}{2}(R_{z,v} + R_{z,x}),$$

$$\tilde{t}_v = t_v + \frac{C_l}{2}(R_{v,v} + R_{v,x}), \quad \tilde{t}_x = t_x + \frac{C_l}{2}(R_{x,v} + R_{x,x}),$$

where $R_{y,v} = R_u + R_v$, $R_{y,x} = R_u$, $R_{z,v} = R_u + R_v$, $R_{z,x} = R_u$, $R_{v,v} = R_u + R_v$, $R_{v,x} = R_u$, and $R_{x,x} = R_u + R_x$.

Now, we compute the r's, the Elmore delays when $C_v = 1$, $C_x = -1$, and all other node capacitances are zero.

$$r_v = R_u(1 + (-1)) + R_v \cdot 1 = R_v, \; r_x = R_u(1 + (-1)) + R_v \cdot (-1) = -R_x,$$
$$r_y = R_u(1 + (-1)) + R_v \cdot 1 + R_y \cdot 0 = R_v, \; r_z = R_u(1 + (-1)) + R_v \cdot 1 + R_z \cdot 0 = R_v.$$

The computation of the new Elmore delays after link insertion is now trivial:

$$\hat{t}_y = \tilde{t}_y - \frac{\tilde{t}_v - \tilde{t}_x}{R_l + r_v - r_x} r_y, \quad \hat{t}_z = \tilde{t}_z - \frac{\tilde{t}_v - \tilde{t}_x}{R_l + r_v - r_x} r_z$$

$$\hat{t}_x = \tilde{t}_x - \frac{\tilde{t}_v - \tilde{t}_x}{R_l + r_v - r_x} r_x, \quad \hat{t}_v = \tilde{t}_v - \frac{\tilde{t}_v - \tilde{t}_x}{R_l + r_v - r_x} r_v$$

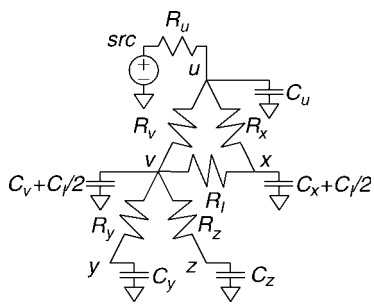


FIGURE 13.52

Insertion of a cross-link to the RC tree in Figure 13.17.

Now, we examine the effects of the additions of capacitors and resistor on the skew between u and w. The new skew between u and w, denoted as $\widehat{skew}_{u,w}$, is

$$\widehat{skew}_{u,w} = \frac{R_l}{R_l + r_u - r_w}\left(skew_{u,w} + \frac{C_l}{2}\left(R_{u,u} - R_{w,w}\right)\right)$$

It is clear that the addition of C_l almost always results in a new skew between u and w as $R_{u,u}$ is almost certainly different from $R_{w,w}$. If we neglect C_l from the preceding equation, we may make the following observations:

1. The addition of R_l reduces the magnitude of the skew between u and w (*i.e.*, $|\widehat{skew}_{u,w}| < |skew_{u,w}|$). In other words, the skew variability of u and w is reduced. This stems from the fact that $r_u = R_{u,u} - R_{u,w}$ and $r_w = R_{u,w} - R_{w,w}$ are the Elmore delays obtained by setting $C_u = 1$ and $C_w = -1$ and zeroing all other node capacitances.
2. For the special case in which $skew_{u,w} = 0$, the skew remains zero after the addition of R_l.
3. The farther u and w are from their youngest (nearest) common ancestor, the smaller the skew variability.

These observations motivate the following procedure for link insertion [Rajaram 2004] in which links are inserted only between nodes that have zero skews (Observation 2). Moreover, only sink pairs are considered (Observation 3). To overcome the effects caused by the link capacitance, the clock tree is tuned after we divide the link capacitance and add them to the two nodes. The purpose of the tuning is to restore the original skews of all sink pairs in the tree. This can be accomplished easily by applying the DME algorithm. Consequently, the addition of link resistance will either maintain the zero skew between the two nodes or reduce the skew variability of the two nodes. Because the tuning process can be expensive, all link capacitances are inserted simultaneously, and the tuning is performed only once.

The remaining problem is that of selecting the set of sink pairs for link insertion. In the approach adopted in [Rajaram 2004], only sink pairs that satisfy the following three selection rules simultaneously are selected:

1. Observation 1 suggests that the smaller the link resistance to loop resistance ratio, $\frac{R_l}{R_l + r_u - r_w}$, the lower the skew variability. Thus, sinks u and w may be selected only when the ratio is no greater than a user-specified threshold $\alpha_{\max}$.
2. The effect of the link capacitance on skew is small when $\beta = |\frac{C_l}{2}(R_{u,u} - R_{w,w})|$ is small. Hence, a link may be inserted between sinks u and w only when β is no greater than a user-specified threshold $\beta_{\max}$. This rule has the effect of selecting two sinks that are in close proximity and have similar path lengths from the source.
3. Observation 3 suggests that the level of the youngest common ancestor (YCA) for a sink pair should not be too high. Here, the level of a node

refers to the number of edges between the source and the node. Consequently, a sink pair may be selected only when the level of its YCA is no greater than a user-specified threshold $\gamma_{\max}$.

Other link insertion algorithms can be found in [Rajaram 2004, 2005, 2006].

Example 13.5 Applying Cross-Link Selection Rules: In Figure 13.51, the level of *p* and the YCA of *u* and *w* is 2. Assuming that $\gamma_{max} = 3$ and that the first two selection rules are also satisfied, the cross-link connecting *u* and *w* is inserted as shown.

Although the level of the YCA of *u* and *v* is $1 \leq \gamma_{max}$, the cross-link between *v* and *v* is not inserted, because *u* and *v* are too far apart and the link resistance to loop resistance ratio is too high.

We also should not insert a cross-link between *x* and *y*, because the level of their YCA is greater than γ_{max}.

13.4 POWER/GROUND NETWORK DESIGN

Now, we shall turn our attention to the analysis and synthesis of ***power/ground*** (P/G) networks. In particular, we focus on the main design challenge highlighted in Section 13.2, namely, *IR* and $L \cdot di/dt$ power supply noise. Some techniques to suppress power supply noise, such as wire sizing, also mitigate the electromigration effects. We follow a similar organization as in the preceding section on clock network design. First, we describe some typical P/G topologies used in designs. Next, we present a random-walk method for the efficient analysis of P/G networks. Last, we focus on the automated synthesis of P/G networks.

13.4.1 Typical power/ground topologies

The design of P/G distribution networks begins with the construction of an appropriate routing topology. First, we consider a simple power distribution scheme that consists of two large concentric rings (one power and one ground) from which comblike structures can be attached (see Figure 13.53). In particular, each comblike structure is commonly used for standard-cell designs. To counter power supply noise and electromigration, the concentric rings, which are connected to V_{DD} and *GND* (or V_{SS}) pads, are typically of a larger width. Similarly, the trunk of a comb is wider than the fingers. Sometimes, the presence of larger modules such as memory blocks or bus lines would destroy the regularity shown in Figure 13.53.

Tree and mesh structures are the most common topologies for power and ground routing. Although the design of a power tree structure (see Figure 13.54)

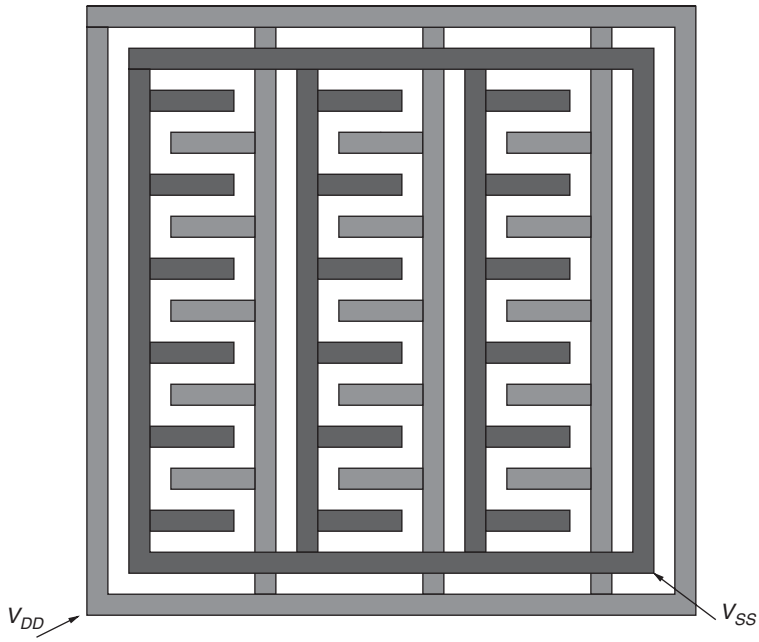


FIGURE 13.53

Interleaved power and ground routing for standard cell designs.

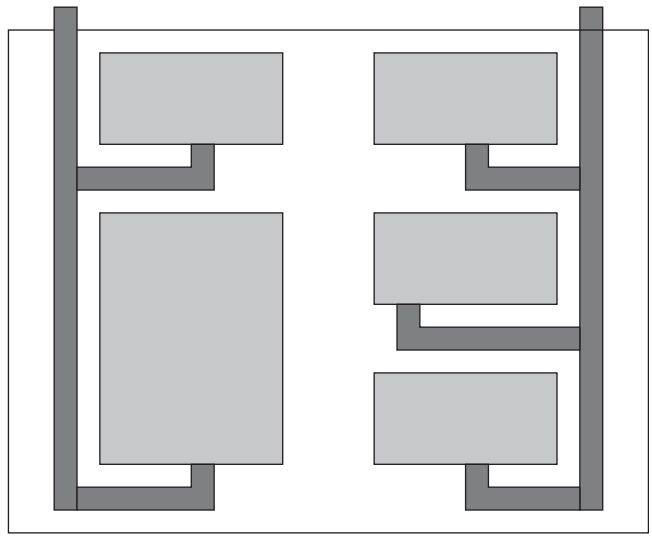


FIGURE 13.54

Local tree–based power distribution technique.

offers efficiency for resource consumption, mesh structures perform better in minimizing voltage and current variations in the supply networks. A mesh structure consists of a rectangular grid of orthogonal wires spanning the whole circuit; see Figure 13.55 for an example of a generic mesh structure. In modern-day microprocessors, the wires of the grid extend across several layers [Singh 2005]. Wires in different layers are connected through vias, solid black blocks joining two metal layers differentiated by shade, as seen in Figure 13.55.

As in the case of clock network design, it is common to see a combination of these various topologies in a single P/G network. Because of its robustness, a mesh structure typically sits at the topmost level in the hierarchy of a P/G network. Comblike structures and tree topologies, with wire width tapering, are usually used for the local distribution of power supply, because they are more frugal on routing resources.

Packaging technologies also play a significant role in enhancing the robustness of power supply. One of the most common interfaces for external power being supplied to an IC is along the periphery of the die (see Figure 13.56).

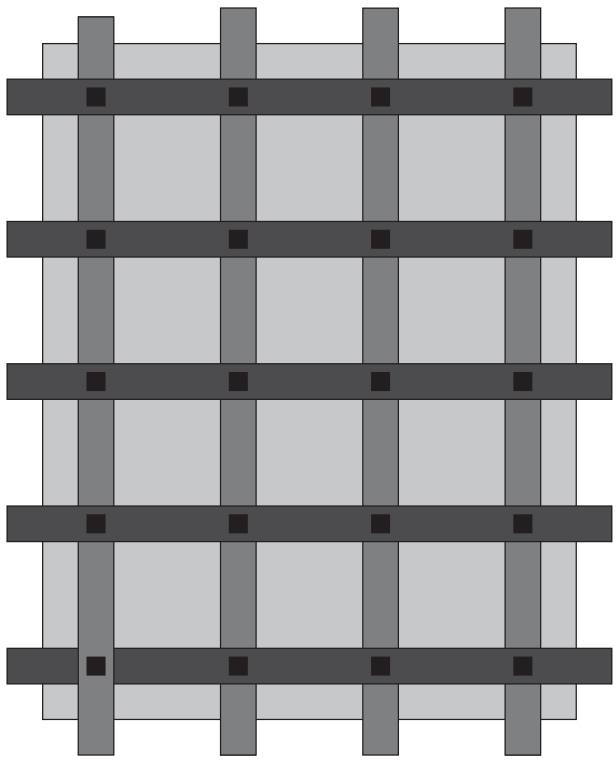


FIGURE 13.55

Typical power mesh structure.

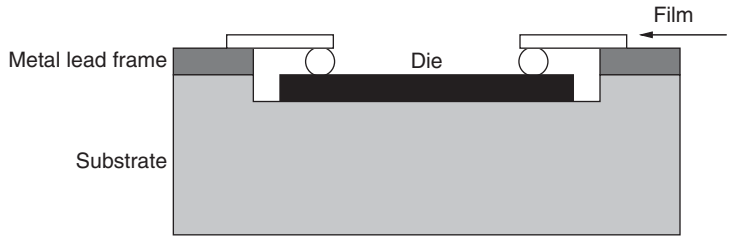


FIGURE 13.56

Use of polymer film wires to connect power to periphery of chip area.

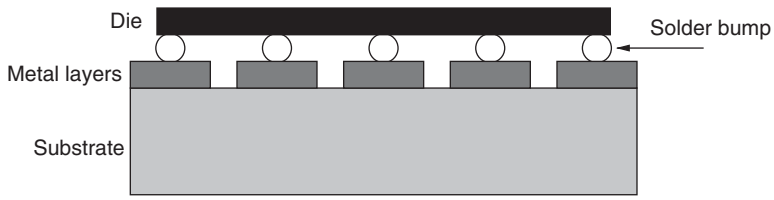


FIGURE 13.57

Flip-chip design to distribute power to any location on chip area.

Such packaging technology is typically used with comblike and tree topologies. In the comblike topology for example, the V_{DD} and *GND* pads on the periphery of the die can be easily connected to any point on the concentric rings.

Flip-chip packaging makes it possible to supply external power into the interior of the die area directly (see Figure 13.57). For flip-chip mounting, V_{DD} and *GND* pads are distributed across the topmost layer. The die is flipped upside down and connected to the substrate of the package with solder bumps. Flip-chip packaging provides two benefits: the power supply is available at any position on the chip and the parasitic inductances and capacitances of such packages are lower. Used in conjunction with a power mesh, the V_{DD} (or *GND*) pads usually reside on the grid points of the mesh. The pad density of a region of the chip is a function of the current demand in that region. Note that flip-chip packaging can also help address the clock distribution problems.

For power minimization, dual- or multiple-V_{DD} designs have become quite popular in recent years. For such a design, high V_{DD} is used for high-performance components, and low V_{DD} powers low-performance components to conserve energy. For a dual-V_{DD} design, for example, three P/G networks are required: V_{DD}^H (the high supply voltage), V_{DD}^L (the low supply voltage), and *GND*. Both V_{DD}^H and V_{DD}^L could be supplied externally. Alternately, an on-chip voltage regulator can be used to take a single externally supplied voltage $V_{DD}^H = V_{DD}$ and drop it to supply V_{DD}^L.

13.4.2 **Power/ground network analysis**

P/G network analysis can involve either the DC or the transient analysis of the network. DC analysis is required for finding the static *IR* drop or finding steady state values corresponding to the *IR* drop caused by the average current flowing through a power supply network. Transient analysis is concerned with determining the effects of switching activity, which is essentially finding voltage fluctuations on a P/G network. This is required for determining the dynamic *IR* drop and $L \cdot di/dt$ noise.

IR drop analysis of a power supply network can be done efficiently, because the resistance and capacitance matrices are both positive definite. Iterative methods, such as conjugate gradient, can be used for efficiently simulating large power distribution networks [Lin 2001]. The analysis of $L \cdot di/dt$ noise is more difficult, because including the inductance matrix in the modified nodal analysis formulation results in matrices that are not positive definite. Specifically, the use of fast iterative methods directly becomes difficult [Lin 2001]. However, by reverting to the nodal formulation, matrices that are symmetric positive definite [Chen 2001] can be formulated, and the inherent structure/conditioning can be exploited. As in the case of clock network analysis, we will focus on the modeling and analysis of P/G networks that are composed of only resistive and capacitive elements.

Traditionally, P/G networks have been modeled with large linear time-invariant models. The supply lines are modeled as distributed RC segments. For a present-day supply network, such a model can consist of millions of nodes and segments. The sources that supply power are modeled as voltage sources, whereas the drain elements that draw currents can be modeled as time-varying current sources. Figure 13.58 shows the models for power sources and drains. The Modified Nodal Analysis (MNA) of such models yields equations of the form

$$Gx + C\dot{x} = b \tag{13.13}$$

where x is the vector of node voltages, b is the vector of current sources, G is the conductance matrix, and C is the admittance matrix, which consists of capacitive elements. We assume that voltage sources have been transformed into Norton equivalent circuits. Hence, we consider only current sources in b. For an RLC network, x and C would also contain inductor currents and inductive elements, respectively.

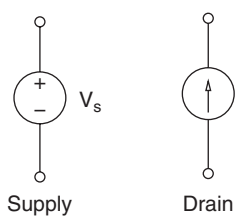


FIGURE 13.58

Models for supply and drain in a P/G network.

Applying the standard trapezoidal integration scheme in Equation (13.13) with step size h results in the following linear system of equations:

$$\underbrace{\left(G + \frac{2C}{h}\right)}_{A} x^{k+1} = \underbrace{b^{k+1} + b^{k} - \left(G - \frac{2C}{h}\right)x^{k}}_{b} \tag{13.14}$$

where x^k is the vector of node voltages at time $k \cdot h$ and (*i.e.*, the kth time step).

The solution of Equation (13.14) involves an inversion of the coefficient matrix A. As more devices are packed on a single chip, the size of the power distribution network will increase, and consequently the matrix A tends to be of very large size. A direct solve of Equation (13.14) hence becomes impractical because of the large amount of memory and computation required. However, networks such as mesh and tree will correspond to a matrix A that is sparse and structured. In addition, A is diagonally dominant and symmetric [Kozhaya 2001]. A number of methods have been developed, that exploit these properties of A for analysis and optimization of P/G networks.

We will begin by discussing the specific sparsity structure of the matrix A that arises in the analysis of RC mesh structures. Consider a simple example of a 3×3 power mesh, as shown in Figure 13.59. The sparsity structure of matrix A for this example is as follows:

$$\begin{pmatrix}
\times & \times & & \times & & & & & \\
\times & \times & \times & & \times & & & & \\
 & \times & \times & & & \times & & & \\
\times & & & \times & \times & & \times & & \\
 & \times & & \times & \times & \times & & \times & \\
 & & \times & & \times & \times & & & \times \\
 & & & \times & & & \times & \times & \\
 & & & & \times & & \times & \times & \times \\
 & & & & & \times & & \times & \times
\end{pmatrix}$$

A multigrid-like technique was proposed in [Kozhaya 2001, 2002]. In this approach, the original large system is first mapped to a coarse grid of reduced size. The reduced system is solved, and the solution is then mapped back to the original system through interpolation. A method for hierarchical analysis

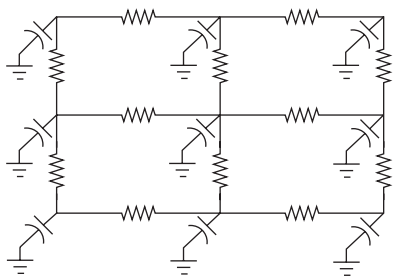


FIGURE 13.59

A 3×3 RC mesh structure.

of power distribution networks was provided in [Zhao 2002]. The power grid is first divided into a global grid and several local grids. Macromodels for the local grids are then generated by use of efficient numerical methods. These macromodels can then be used for simulating the global grid. In addition, several techniques based on iterative methods that use sparse linear system techniques have been proposed as well. Techniques based on preconditioned Krylov subspace methods and successive overrelaxation techniques were proposed in [Chen 2001] and [Zhong 2005], respectively.

Finally, we will examine in more detail a stochastic technique based on the random walk method [Qian 2003]. Here the problem of simulating an RC power mesh is translated into a stochastic game that will proceed iteratively until some measure of convergence is achieved. To motivate the procedure, we begin by examining a simple situation where we have a cross-shaped structure created by four resistors with a current source at the central node, as shown in Figure 13.60. The current at the central node x is described through KCL:

$$\sum_{i=1}^{4} g_i(V_i - V_x) = I_x$$

where g_i and V_i are the conductance and voltage for each bordering node, respectively. Thus, we can solve for the voltage at the central node:

$$V_x = \sum_{i=1}^{4} \frac{g_i}{\sum_{j=1}^{4} g_j} V_i - \frac{I_x}{\sum_{j=1}^{4} g_j} \tag{13.15}$$

If we construct Equation (13.15) for each node in a resistive network and solve the linear equations simultaneously, we arrive at the exact voltage values across the network. Instead, let us form a basic stochastic game that mimics the problem of solving for these voltages.

Imagine a traveler who is currently at a position x (see Figure 13.61). With some probability $p_{x,i}$, the traveler will decide to walk down one of the available roads i. The sum of the probabilities for all of the roads that intersect at x is 1, but the individual probabilities have yet to be determined. Each node that is not labeled "HOME" is considered a hotel that must be visited (with a cost incurred, denoted as m_x), and the traveler must continue along until a final home location

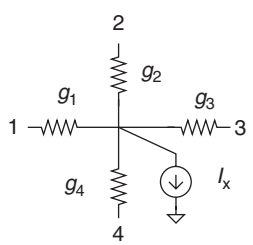


FIGURE 13.60

Portion of a resistive network.

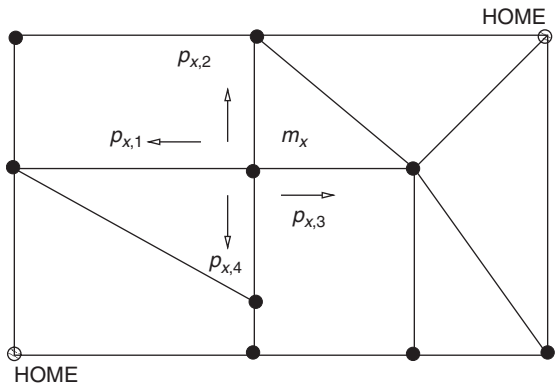


FIGURE 13.61

Random walk game.

is reached (and fixed prize amount is awarded, denoted as m_0). Thus, we can define a gain function f for the amount of money earned on the walk given that we began at a non-home node x that has n bordering locations labeled $1, 2, \ldots, n$:

$$f(x) = \sum_{i=1}^{n} p_{x,i} f(i) - m_x$$

Given that a traveler will remain at the home location once it is reached, we conclude that $f(y) = m_0$ for all home nodes y. An analogy can now be drawn between the resistive V_{DD} network and this stochastic game:

$$p_{x,i} = \frac{g_i}{\sum_{j=1}^{n} g_j}, \quad m_x = \frac{I_x}{\sum_{j=1}^{n} g_j}, \quad m_0 = V_{DD}$$

Repeated iterations of the random walk game will give the values of the voltages for each node $f(x) = V_x$ in the resistive network. To apply this technique to RC power network analysis, we have to replace each capacitor in the RC network by a resistor and a voltage-controlled current source. This simple extension is further developed in the section exercises.

13.4.3 Power/ground network synthesis

The synthesis of a P/G network involves the determination of its topology, the placement of power pads, the appropriate sizing of power lines, and the insertion of decoupling capacitances. When designing a power distribution network, it is important that both the maximum allowable current density for each wire and the maximum voltage drop at each node are within specifications. The maximum current density constraints must be satisfied to avoid the wearing out of

metal wires, whereas the voltage drop criteria are required for maintaining correct functioning of the IC. Meanwhile, it is important that the wiring resource consumption of a P/G network is minimized.

13.4.3.1 *Topology optimization*

A variety of techniques have been proposed for topology optimization of P/G supply networks [Rothermel 1981; Syed 1982; Xiong 1986; Mitsuhashi 1992; Singh 2004, 2005]. For modern dense circuit designs, power grids can involve millions of wires, and it becomes necessary to formulate fast techniques for the design of such networks.

The synthesis technique in [Singh 2005] is based on the recursive partitioning of the chip area. The procedure starts with assigning a very coarse grid for the whole design, assuming very wide wires and large pitch. The grid is then recursively bipartitioned. The coarse grid in each of the two partitions is refined such that each wide wire is replaced by several narrower wires with smaller pitch.

The amount of computation needed to perform the analysis after each refinement step does not grow substantially as a result of the use of locality (*i.e.*, independently solving only a small local grid in each iteration). The procedure halts when the grid topology is able to satisfy the two specifications described previously.

13.4.3.2 *Power pad assignment*

We will now examine the problem of determining the optimal number of power pads and their locations when dealing with a grid topology. The objective is to minimize the number of power pads necessary to meet the two design considerations discussed earlier. A ***mixed integer linear program*** (MILP) formulation is given in [Zhao 2004]. The MILP formulation involves a set of potential power pad locations, *PC*, and a set of observation points, *OP*, on the power grid. By specifying a subset of the total number of nodes as observation points, a macromodel can be formed for the power grid system, as shown in Figure 13.62. The governing dynamics of this system can be described by the following equation:

$$I = A \cdot V + S \tag{13.16}$$

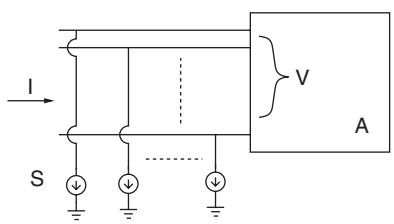


FIGURE 13.62

Macromodel schematic.

where I is a vector of currents flowing into the model through the ports, A is the conductance matrix, and V is the port voltage vector. By introducing 0-1 integer variables, we can construct constraints for the current seen at any of the potential pad locations:

$$\begin{array}{rcl} V_i - V_{DD} \cdot z_i & \geq & 0 \\ V_i & \leq & V_{DD} \\ V_i & \geq & V_t \\ I_t \cdot z_i - I_i & \geq & 0 \\ I_i & \geq & 0 \end{array} \tag{13.17}$$

Here, I_t is the maximum current allowed through a pad, V_t is the minimum voltage for any node in the power grid, and z_i is a 0-1 variable designating if location i is occupied by a pad. Thus, we can formulate the following optimization problem:

$$\begin{array}{rl} \min & \sum_{i \in PC} z_i \quad i \in \{0,1\} \\ \text{subject to} & V_j \geq V_t \quad j \in OP \\ & I_i \text{ and } V_j \text{ satisfy Equation (13.16) for } j \in OP \cup PC \text{ and } i \in PC \\ & I_i \text{ and } V_j \text{ satisfy Equation (13.17) for } i \in PC \end{array}$$

A solution based on a branch-and-bound algorithm can be used for solving such an optimization problem. Since the MILP solution procedure can become highly expensive for a large number of variables, which in this case are the power pad locations, a heuristic can be used to aid in reducing this run time complexity. In [Zhao 2004], for example, a divide-and-conquer approach is used to divide the whole power grid into several partitions and then assign pads for each partition independently.

The heuristic presented in [Oh 1998] simultaneously performs pad assignment and P/G routing. In this approach, an attempt is made to evenly distribute the inductively induced voltage fluctuation across the pads, while minimizing routing of single-pad trees.

13.4.3.3 *Wire width optimization*

We now discuss for a given topology how the wire widths can be optimized to meet both maximum allowable voltage drop and physical breakdown constraints [Chowdhury 1985]. By varying the wire width we will be able to control the resistance of the wire and, therefore, the amount of current flowing through that path or route. This, in turn, will allow us to meet any desired maximum allowable voltage drop required by the design specifications. In this case voltage drop will be measured from the actual power pad to a power supply pin of a gate or module. In addition, for the optimization problem a minimum allowable width for each branch in the network will be imposed. This minimum width is enforced to avoid metal migration effects that will affect the physical reliability of the design. Specifically, if the cross-sectional area increases or the current density decreases, we can assume that the average lifetime of the wire will be longer. Finally, our overall goal (objective function) will be to minimize the total area required for the routing of the power network. This is equivalent to minimizing a weighted sum of the wire width needed to meet the specifications for the network.

This or similar optimization problems can be formulated as mathematical programs, which are then solved numerically. If we consider the formulation shown in [Mitsuhashi 1992] the result is a nonlinear programming approach for solving the problem. Here the possible wire widths are some discrete values based on a realistic fabrication situation. The objective function can be described as follows:

$$A = \sum_{i=1}^{n} w_i l_i = \sum_{i=1}^{n} \frac{|\rho I_i l_i^2|}{x_i}$$

where w_i and l_i are the width and length, respectively, for the n branches of the power network. The term I_i is the maximum current flow across a branch with resistivity ρ, and the voltage drop is denoted by x_i. A maximum allowable voltage drop v_j across any path p_j can be assured through the following constraints:

$$\sum_{i \in p_j} x_i \leq \Delta v_j$$

The minimum width W allowed by the fabrication process can be incorporated through these n conditions:

$$w_i = \frac{\rho I_i l_i}{x_i} \geq W$$

Finally, by putting constraints on the maximum current-wire width ratio, we can avoid the problems caused by metal migration:

$$\frac{I_i}{w_i} = \frac{x_i}{\rho l_i} \leq \sigma_i$$

where σ_i is the maximum allowable current density across branch i.

With an alternate mathematical model, a linear formulation of the optimization problem given in [Tan 2003] facilitates the use of more efficient methods of solution. Furthermore, interested readers can examine [Luenberger 2003], which provides detailed explanations of numerical schemes used to solve the optimization problems described previously.

13.4.3.4 *Decoupling capacitance*

One of the most powerful techniques for reducing power supply noise caused by *IR* drop and $L \cdot di/dt$ noise is to use decoupling capacitors across the power grid [Rao 2001]. It is commonly understood that charge is required to energize a load capacitor, where the charge in this case is supplied by the current flowing through the power supply network. However, such a mechanism introduces power supply noise caused by the intrinsic wire resistance and inductance in the network.

Alternately, charge can also be supplied by another capacitor, which can be placed in close proximity to the load capacitance. Because this additional capacitance is located near the load, it will reduce the overall size of the current loop and act to supply charge to the load. This situation is illustrated in Figure 13.63.

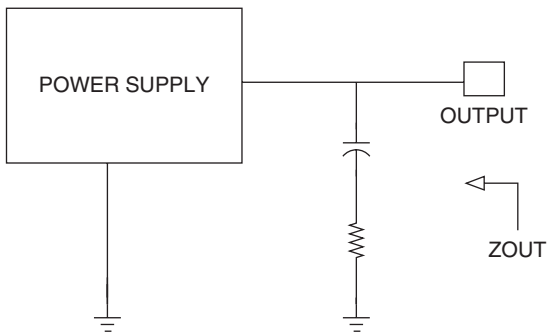


FIGURE 13.63

Decoupling network.

Consequently, current does not need to flow across most of the parasitics for the power supply network, and the current-induced noise is reduced. The needed action for a decoupling capacitor can be efficiently implemented with the gate capacitance of a transistor. With increasing noise values, it is often necessary to allocate as much as 10% of the total chip area to decoupling capacitors. Therefore, it is crucial to estimate the size and area needed for decoupling capacitor assignment.

There have been many studies into decoupling capacitor placement and optimization. First, we address the problem of estimating the amount of decoupling capacitor that each circuit module will require. The amount of decoupling capacitor required by each circuit module at the floorplanning level can be calculated as follows: suppose the maximum voltage noise that a module can tolerate is V_{noise}^{lim}. Let $Q = \int_0^{\tau} I(t)dt$ denote the maximum total charge that this module will draw from the power supply over τ, the duration that the current waveform $I(t)$ lasts. A greedy scheme for decoupling capacitor estimation is to allocate $C = Q/V_{\text{noise}}^{lim}$ to this module.

However, this scheme might result in overallocation of decoupling capacitors, because each decoupling capacitor in practice can be shared by several modules [Zhao 2001]. An iterative approach was proposed instead. An initial estimate of decoupling capacitor required for each module is expressed as

$$\theta = \max\left(1, \frac{V_{\text{noise}}}{V_{\text{noise}}^{lim}}\right), \quad C = \frac{\left(1 - \frac{1}{\theta}\right)Q}{V_{\text{noise}}^{lim}}$$

where V_{noise} denotes the power supply noise that the module experiences. After C is added to the module, the power noise, V_{noise}, can be recalculated for all the other modules, and the remaining C are allocated accordingly. Because the decoupling capacitor at a module also helps in reducing the power supply noise at a neighboring module, the amount of decoupling capacitor allocated with this procedure can be significantly smaller than that of the greedy scheme.

Several techniques have been proposed for decoupling capacitor budgeting at the floorplanning level [Su 2003]. An iterative procedure involving circuit simulation and floorplanning was proposed in [Chen 1997]. Here, a circuit simulator is used to first analyze the switching noise and determine any hot spots (locations of excessive voltage drop). Then, the sizes of the decoupling capacitors needed to meet voltage drop specifications are determined. On-chip decoupling capacitors are usually implemented as MOS capacitors, which has a unit area capacitance of $C_{ox} = \varepsilon_{ox}/t_{ox}$, where t_{ox} is the oxide thickness and ε_{ox} is the permittivity of SiO_2. Given a decoupling capacitance C, the silicon area required for the fabrication of the decoupling capacitor is $S = C/C_{ox}$.

After this initial circuit simulation stage, the floorplanner calculates the area required to implement these decoupling capacitors, as well as possible locations for their placement. This information can then be sent back to the circuit simulator, which simulates the activity on the new floorplan with the additional added decoupling capacitors. The iterative procedure continues until the power noise is suppressed to within the required limits.

A linear programming–based technique to allocate white space for decoupling capacitors has been proposed in [Zhao 2001]. White space for decoupling capacitor placement is allocated in two stages. The first stage involves allocation of existing white space with a linear program. The objective here is to maximize the utilization of the existing white space. Suppose the white space in the existing floorplan has been partitioned into H rectilinear blocks, called white-space modules. For each white-space module w_k, we use A_k to denote its area and N_{w_k} to denote the set of circuit modules that are adjacent to it. Assume that there are M circuit modules. Let $S^{(j)}$ denote the silicon area required for the decoupling capacitor allocated to circuit module m_j for power supply noise suppression. We use N_{m_j} to denote the set of white-space modules that are adjacent to circuit module m_j. We show a floorplan of 7 rectangular circuit modules and 3 rectilinear white-space modules in Figure 13.64. In this example, $N_{w_1} = \{m_3, m_4, m_5, m_6\}$ and $N_{m_1} = \{w_3\}$.

Let $x_k^{(j)}$ be the amount of white space allocated to circuit module m_j from an adjacent white-space module w_k. The white space utilization problem can be formulated as follows:

$$\begin{aligned}
\max \quad & \sum_{k=1}^{H} \sum_{j \in N_{w_k}} x_k^{(j)}, & & & \\
\text{subject to} \quad & \sum_{j \in N_{w_k}} x_k^{(j)} & \leq & A_k, & k = 1, 2, \ldots, H \\
& \sum_{k \in N_{m_j}} x_k^{(j)} & \leq & S^{(j)}, & j = 1, 2, \ldots, M \\
& x_k^{(j)} & \geq & 0, & \forall k, \forall j \in N_{w_k}
\end{aligned}$$

In the second step, the floorplan is stretched, if necessary, such that extra white space can be created to meet all the decoupling capacitor requirements of the circuit.

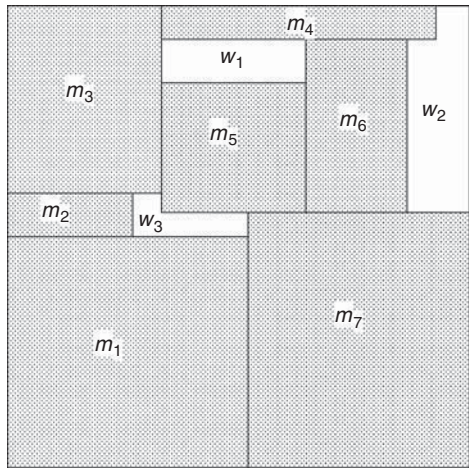


FIGURE 13.64

A floorplan of 7 circuit modules and 3 white-space modules.

All the decoupling capacitor allocation methods require the estimation of the current waveforms that result in hot spots. It is often difficult to accurately estimate the current waveforms in general situations. Hence, the allocation of decoupling capacitor is usually based on the worst-case voltage drops, resulting in an overallocation of resources for decoupling capacitors. To alleviate this problem, several sensitivity-based techniques for decoupling capacitor optimization have been proposed [Bai 2000; Su 2000; Fu 2004].

We now show how the sensitivity of a node voltage with respect to decoupling capacitances can be calculated. In [Su 2003], the authors used the adjoint network method [Director 1969] to calculate critical nodes that can benefit the most from introducing decoupling capacitance. First, an estimate of the current-induced noise at each node in the grid is constructed on the basis of both the possible tunable circuit parameters for the decoupling capacitor and the voltage drop observed at the node.[3] Then, an adjoint circuit is obtained by shorting all voltage sources and opening all current sources in the original power network. Finally, for each node with a non-zero noise measurement, appropriate current sources are applied. This allows for a measurement of the sensitivity to be computed:

$$\frac{\partial Z}{\partial C} = \int_0^T \psi_c(T-t)\dot{v}_c(t)dt \tag{13.18}$$

[3]As the circuits considered in [Su 2003] are standard cell designs, the height of the decoupling capacitors are constrained to be the same as the height of standard cells. Hence, the only tunable circuit parameters are the widths of the decoupling capacitors.

where Z is the sum of the current-induced noise estimates, $\psi_c(\tau)$ is the waveform across the capacitor C in the adjoint circuit, and $v_c(t)$ is the voltage waveform from the original power-grid analysis. This information can then be used to formulate a quadratic program for determining the optimal decoupling capacitor allocation area.

13.5 CONCLUDING REMARKS

There is a rich body of work in the general areas of automated modeling and synthesis of clock distribution networks and P/G delivery systems. This chapter focuses mainly on the modeling and synthesis of clock and power/ground networks composed of RC elements. There are many existing studies that address the problem of synthesizing such networks based on more complete models that also consider inductive parasitics. These studies typically adapt many existing analysis and synthesis techniques covered in this chapter, with varying degrees of success, to address the challenge of incorporating inductive elements in the networks. Parametric variations add further challenges along a different dimension. A bigger challenge is the comodeling and cosynthesis of clock and P/G networks. As the design margin gets smaller, it is getting harder to ignore the interplay of these two large networks.

13.6 EXERCISES

13.1. (Delay Modeling) Let R_{ij} denote the resistance of the common path $Path(src, i) \cap Path(src, j)$ from the source *src* to nodes *i* and *j* in an RC tree T. Besides the Elmore delay, Rubinstein, Penfield, and Horowitz [Rubinstein 1983] also defined the following two delay models:

$$\begin{aligned} t_p &= \sum_{k \in T} R_{kk} C_k \\ t_i &= \left(\sum_{k \in T} R_{ki}^2 C_k \right) / R_{ii} \end{aligned}$$

Although there is only one t_p for a given RC tree T, each node i in T has an associated t_i. It can be shown that $t_i \leq t_{Elmore,i} \leq t_p$.

(a) Write down the pseudo-code to compute t_p in linear time.

(b) Write down the pseudo-code to compute all t_i's in linear time.

13.2. (Skew Scheduling) Consider a constraint graph $G_C(V, E)$ obtained from a set of skew constraints $\mathcal{C}$. Suppose G_C has no negative cycles. Let t_i be the clock delay assigned to clock pin s_i. Suppose the given schedule is not feasible, which implies that for some edge $e_{j,i} \in E$,

$$t_i > t_j + w_{j,i}$$

When this occurs, we apply the following relaxation to change the clock delay assigned to clock pin s_i:

$$t_i \leftarrow t_j + w_{j,i}$$

(a) Show that the process of changing the clock delays assigned to clock pins with iterations of the relaxation step will converge to a feasible clock schedule.

(b) What is the worst-case time complexity for the algorithm to converge to a feasible clock schedule?

13.3. (Zero-Skew Routing) Given two Manhattan arcs, write down the pseudo-code to compute the distance between them.

13.4. (Zero-Skew Routing) Consider two merging segments ms_a and ms_b that are of distance L apart. Suppose they are merged at parent node v, with non-negative wire lengths $|e_a|$ and $|e_b|$, as computed in Section 13.3.3.3. Write down the pseudo-code to compute ms_v, the merging segment at v.

13.5. (Bounded-Skew Routing) Consider two merging regions that are polygons formed by rectilinear line segments and Manhattan arcs, as shown in Figure 13.33, for bounded-skew routing. Write down the pseudo-code to compute the distance L between them.

13.6. (Bounded-Skew Routing) In Section 13.3.3.4, we stated that for bounded-skew routing, the global skew on a boundary segment of a merging region that is a Manhattan arc is a constant.

(a) For the path length delay model, how would the global skew change along a rectilinear boundary segment?

(b) For the Elmore delay model, how would the global skew change along a rectilinear boundary segment?

13.7. (Bounded-Skew Routing) You are given a routine called Compute_merging_region, which takes in two merging regions mr_a and mr_b as input, and outputs a merging region mr_v as a result of performing a bounded-skew merge operation on mr_a and mr_b. Suppose the time complexity of this routine is $O(1)$. Write down the pseudo-code to compute mr_e for all edges $e \in G$ for a given abstract topology G. Recall that mr_e refers to the merging region if the root node of G is relocated to e (see Section 13.3.3.4). Assume that you already have the merging regions for all nodes in G. The time complexity of your algorithm should be linear.

13.8. (Bounded-Skew Routing) Use the bucket decomposition in Section 13.3.3.3 in the Greedy-BST/DME algorithm. If you also incorporate the re-rooting algorithm in Exercise 13.7, what is the time complexity of your Greedy-BST/DME algorithm?

13.9. (Useful-Skew Routing) Suppose you have been given a skew schedule. In other words, all skew constraints are equality constraints.

Modify the zero-skew routing tree algorithm in Algorithm 13.4 to perform prescribed useful-skew routing.

13.10. (Buffer Insertion) In Section 13.3.4.1, the merging region mr_{v_a} is computed by computing the merging segments V_a and V'_a, assuming that a buffer is inserted at v and a, respectively. Let C_{T_a} and C_{T_b} denote the capacitances of the DC-connected subtrees rooted at a and b, respectively. Also, let t_a and t_b denote the sink delays in the two subtrees. Let L denote the distance between a and b.

(a) Write down the respective equations for the distances of V_a and V'_a from a.

(b) Consider a sampling segment s in mr_{v_a}. Let d_s denote the distance of s from a. Determine $|e_a|$ and the location of the buffer along the interconnect from v to a.

13.11. (Clock Gating) In Section 13.3.4.2, we demonstrated how $C_{T_x}^{tot}$, GC_{T_x}, $GC_{T_x}^{eff}$, UGC_{T_x} and $C_{T_x}^{eff}$ can be updated. However, we ignore the parasitic capacitance T_{cg}/R_{cg} in the process.

(a) Write down the steps to update these capacitances, taking into account T_{cg}/R_{cg}.

(b) Suppose instead of a clock gate, we insert a buffer. How do you update these capacitances?

13.12. (Wire Sizing) In Section 13.3.4.3, we use wire sizing to reduce the delay caused by a wire. We will investigate the possibility of the use of wire sizing to slow down a wire in this problem. Consider a wire with 4 segments, with each segment modeled as a lumped RC element. For simplicity, we assume an L-type circuit for the lumped RC element (*i.e.*, the capacitance is at the downstream endpoint of the resistance). Let the width choices form the set $\{1, 2, 3, 4, 5\}$. A width choice of i has a wire resistance of value $1/i$ and a wire capacitance of value i. Determine a wire width assignment that would result in the longest wire delay. You may want to write a program to enumerate all possible assignments.

13.13. (Cross-Link Insertion) Assuming that every node in a tree has a field to record its distance (*i.e.*, number of edges) from the root node (see the definition of the level of a node in Section 13.3.4.4). Write down an algorithm to determine the youngest common ancestor (YCA) of two nodes without visiting any edges not on the paths from YCA to the two nodes.

13.14. (Random Walk) Starting from the MNA equations

$$Gx + C\dot{x} = b$$

if a backward Euler approximation (step size h) is assumed we arrive at an equation of the form:

$$\left(G + \frac{C}{h}\right)x^k = b^k + \frac{C}{h}x^{k-1}$$

(a) With the information provided in section 13.4.2 for RC networks, rewrite this equation for a node x only in terms of conductances g_i, current load I_x, and voltage values V_i.

(b) With the preceding equation, formulate the corresponding stochastic game and describe its meaning. (Hint: first reformulate to solve for V_x on the LHS and then describe what each term on the RHS would represent.)

13.15. (Decoupling Capacitor) Consider the floorplan shown in Figure 13.64. Suppose after solving the linear program, you realize that the decoupling capacitance requirement of m_1 cannot be fulfilled. However, there are some slacks in w_1 and w_2. How would you meet the decoupling capacitance requirement of m_1 without increasing the chip area? Suggest an iterative approach on the basis of the linear program in Section 13.4.3.4 to enhance the utilization factor of white space for decoupling capacitor deployment.

13.16. (Decoupling Capacitor) For a decoupling capacitor to be effective in countering the power supply noise induced by the switching activity of a circuit module, the decoupling capacitor must be placed within some distance of the circuit module. Modify the linear program in Section 13.4.3.4 to enforce the constraint on the distance between a decoupling capacitor and the circuit module that it is safeguarding.

ACKNOWLEDGMENTS

We acknowledge Ruilin Wang of Purdue University for his contributions to the sections on "Design Considerations" and "Clock Network Design." We also thank Dr. Aiqun Cao of Synopsys, Professor Jiang Hu of Texas A&M University, Professor Sheldon X.-D. Tan of the University of California at Riverside, and Dr. Chung-Wen Albert Tsao of Cadence Design Systems for reviewing the chapter.

REFERENCES

R13.0 Books

[Bakoglu 1990] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI,* Addison-Wesley, Reading, MA, 1990.

[Cormen 2001] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms,* Second Edition, MIT Press, Cambridge, MA, 2001.

[Luenberger 2003] D. Luenberger, *Linear and Nonlinear Programming,* Kluwer Academic Publishers, Boston, 2003.

[Rabaey 2003] J. M. Rabaey, A. Chandrakasan, and B. Nikolič, *Digital Integrated Circuits: A Design Perspective,* Second Edition, Prentice-Hall, Upper Saddle River, NJ, 2003.

[Rao 2001] T. Rao, *Fundamentals of Microsystems Packaging,* McGraw-Hill, New York, 2001.

R13.2 Design Considerations

[Anderson 2001] C. J. Anderson, J. Petrovick, J. M. Keaty, J. Warnock, G. Nussbaum, J. M. Tendier, C. Carter, S. Chu, J. Clabes, J. DiLullo, P. Dudley, P. Harvey, B. Krauter, J. LeBlanc, P.-F. Lu, B. McCredie, G. Plum, P. J. Restle, S. Runyon, M. Scheuermann, S. Schmidt, J. Wagoner, R. Weiss, S. Weitzel, and B. Zoric, Physical design of a fourth-generation POWER GHz microprocessor, in *Proc. IEEE Int. Solid-State Circuits Conf.,* pp. 232–233, February 2001.

[Black 1969] J. R. Black, Electromigration—A brief survey and some recent results, *IEEE Trans. on Electron Devices*, 16(4), pp. 338–347, April 1969.

[Lin 2001] S. Lin and N. Chang, Challenges in power-ground integrity, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 651–654, November 2001.

R13.3 Clock Network Design

[Anderson 2001] C. J. Anderson, J. Petrovick, J. M. Keaty, J. Warnock, G. Nussbaum, J. M. Tendier, C. Carter, S. Chu, J. Clabes, J. DiLullo, P. Dudley, P. Harvey, B. Krauter, J. LeBlanc, P.-F. Lu, B. McCredie, G. Plum, P. J. Restle, S. Runyon, M. Scheuermann, S. Schmidt, J. Wagoner, R. Weiss, S. Weitzel, and B. Zoric, Physical design of a fourth-generation POWER GHz microprocessor, in *Proc. IEEE Int. Solid-State Circuits Conf.,* pp. 232–233, February 2001.

[Bailey 1998] D. Bailey and B. Benschneider, Clocking design and analysis for a 600-MHz Alpha microprocessor, *IEEE J. on Solid-State Circuits*, 33(11), pp. 1627–1633, November 1998.

[Boese 1992] K. D. Boese and A. B. Kahng, Zero-skew clock routing trees with minimum wirelength, in *Proc. IEEE Int. ASIC Conf.,* pp. 1.1.1–1.1.5, September 1992.

[Boese 1995] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, Near optimal critical sink routing tree constructions, *IEEE Trans. on Computer- Aided Design*, 14(12), pp. 1417–1436, December 1995.

[Chan 1990] P. K. Chan and K. Karplus, Computing signal delay in general RC networks by tree/link partitioning, *IEEE Trans. on Computer-Aided Design*, 9(8), pp. 898–902, August 1990.

[Chao 1992] T.-H. Chao, Y.-C. H. Hsu, and J.-M. Ho, Zero skew clock net routing, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 518–523, June 1992.

[Chao 2008] W.-C. Chao and W.-K. Mak, Low-power gated and buffered clock network construction, *ACM Trans. on Design Automation of Electronic Systems*, 13(1), Article No. 20, pp. 1–20, January 2008.

[Chen 1996] Y. P. Chen and D. F. Wong, An algorithm for zero-skew clock tree routing with buffer insertion, in *Proc. European Design and Test Conf.,* pp. 230–236, March 1996.

[Chen 2002] C. Chen, C. Kang, and M. Sarrafzadeh, Activity-sensitive clock tree construction for low power, in *Proc. Int. Symp. on Low Power Electronics and Design,* pp. 279–282, August 2002.

[Cong 1995a] J. Cong and C.-K. Koh, Minimum-cost bounded-skew clock routing, in *Proc. IEEE Int. Symp. on Circuits and Systems,* pp. 1.215–1.218, April 1995.

[Cong 1995b] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao, Bounded skew clock and Steiner routing under Elmore delay, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 66–71, November 1995.

[Cong 1996] J. Cong and L. He, Optimal wiresizing for interconnects with multiple sources, *ACM Trans. on Design Automation of Electronic Systems*, 1(4), pp. 478–511, October 1996.

[Cong 1998] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao, Bounded-skew clock and Steiner routing, *ACM Trans. on Design Automation of Electronic System*, 3(3), pp. 341–388, July 1998.

[Edahiro 1991] M. Edahiro, Minimum skew and minimum path length routing in VLSI layout design, *NEC Research and Development*, 32(4), pp. 569–575, October 1991.

[Edahiro 1992] M. Edahiro and T. Yoshimura, Minimum path-length equi-distant routing, in *Proc. IEEE Asia-Pacific Conf. on Circuits and Systems,* pp. 41–46, December 1992.

[Edahiro 1993a] M. Edahiro, A clustering-based optimization algorithm in zero skew routing, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 612–616, June 1993.

[Edahiro 1993b] M. Edahiro, Delay minimization for zero-skew routing, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 563–566, November 1993.

[Edahiro 1994] M. Edahiro, An efficient zero-skew routing algorithm, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 375–380, June 1994.

[Elmore 1948] W. C. Elmore, The transient response of damped linear networks with particular regard to wide-band amplifiers, *J. of Applied Physics*, 19(1), pp. 55–63, January 1948.

[Fisher 1982] A. L. Fisher and H. T. Kung, Synchronizing large systolic arrays, in *Proc. SPIE*, 341 pp. 44–52, May 1982.

[Geannopoulos 1998] G. Geannopoulos and X. Dai, An adaptive digital deskewing circuit for clock distribution networks, in *Proc. IEEE Int. Solid-State Circuits Conf.,* pp. 400–401, February 1998.

[Gupta 1997] R. Gupta, B. Tutuianu, and L. T. Pileggi, The Elmore delay as a bound for RC trees with generalized input signals, *IEEE Trans. on Computer- Aided Design*, 16(1), pp. 95–104, January 1997.

[Guthaus 2006] M. R. Guthaus, D. Sylvester, and R. B. Brown, Clock buffer and wire sizing using sequential programming, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 1041–1046, July 2006.

[Huang 1995] J. H. Huang, A. B. Kahng, and C.-W. A. Tsao, On the bounded skew routing tree problem, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 508–513, June 1995.

[Kung 1982] S. Y. Kung and R. J. Gal-Ezer, Synchronous versus asynchronous computation in very large scale integrated VLSI array processors, in *Proc. SPIE*, 341 pp. 53–65, May 1982.

[Kurd 2001] N. A. Kurd, J. S. Barkarullah, R. O. Dizon, T. D. Fletcher, and P. D. Madland, A multi-gigahertz clocking scheme for the Pentium(R) 4 microprocessor, *IEEE J. of Solid-State Circuits*, 36(11), pp. 1647–1653, November 2001.

[Lam 2002] W.-C. D. Lam, C.-K. Koh, and C.-W. A. Tsao, Power supply noise suppression via clock skew scheduling, in *Proc. IEEE Int. Symp. on Quality of Electronic Design,* pp. 355–360, March 2002.

[Lam 2003] W.-C. D. Lam, C.-K. Koh, and C.-W. A. Tsao, Clock scheduling for power supply noise suppression using genetic algorithm with selective gene therapy, in *Proc. IEEE Int. Symp. on Quality of Electronic Design,* pp. 327–332, March 2003.

[Lam 2005] W.-C. D. Lam, J. Jain, C.-K. Koh, V. Balakrishnan, and Y. Chen, Statistical based link insertion for robust clock network design, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 587–590, November 2005.

[Lillis 1995] J. Lillis, C. K. Cheng, and T. T. Y. Lin, Optimal wire sizing and buffer insertion for low power and a generalized delay model, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 138–143, November 1995.

[Lin 1975] H. C. Lin and L. W. Linholm, An optimized output stage for MOS integrated circuits, *IEEE J. of Solid-State Circuits*, SC-10(2), pp. 106–109, April 1975.

[Nagel 1975] L. W. Nagel, SPICE2: A computer program to simulate semiconductor circuits, *Technical Report ERL-M520,* University of California, Berkeley, CA, May 1975.

[Oh 2001] J. Oh and M. Pedram, Gated clock routing for low-power microprocessor design, *IEEE Trans. on Computer-Aided Design*, 20(6), pp. 715–722, June 2001.

[Rajaram 2004] A. Rajaram, J. Hu, and R. Mahapatra, Reducing clock skew variability via cross links, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 18–23, June 2004.

[Rajaram 2005] A. Rajaram, D. Z. Pan, and J. Hu, Improved algorithms for link based non-tree clock networks for skew variability reduction, in *Proc. ACM Int. Symp. on Physical design,* pp. 55–62, April 2005.

[Rajaram 2006] A. Rajaram and D. Z. Pan, Variation tolerant buffered clock network synthesis with cross links, in *Proc. ACM Int. Symp. on Physical design,* pp. 157–164, April 2006.

[Rubinstein 1983] J. Rubinstein, P. Penfield Jr, and M. A. Horowitz, Signal delay in RC tree networks, *IEEE Trans. on Computer-Aided Design*, CAD-2(3), pp. 202–211, July 1983.

[Tam 2000] S. Tam, S. Rusu, U. N. Desai, R. Kim, J. Zhang, and I. Young, Clock generation and distribution for the first IA-64 microprocessor, *IEEE J. of Solid-State Circuits*, 35(11), pp. 1545–1552, November 2000.

[Téllez 1995] G. E. Téllez, A. Farrahi, and M. Sarrafzadeh, Activity-driven clock design for low power circuits, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 62–65, November 1995.

[Tsai 2003] J.-L. Tsai, T.-H. Chen, and C. C.-P. Chen, ε-Optimal minimum delay/area zero-skew clock tree wire-sizing in pseudo-polynomial time, *Proc. ACM Int. Symp. on Physical Design,* pp. 166–173, April 2003.

[Tsao 2002] C.-W. A. Tsao and C.-K. Koh, UST/DME: A clock tree router for general skew constraints, *ACM Trans. on Design Automation of Electronic Systems*, 7(3), pp. 359–379, July 2002.

[Tsay 1991] R.-S. Tsay, Exact zero skew, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 336–339, November 1991.

[van Ginneken 1990] L. P. P. P. van Ginneken, Buffer placement in distributed RC-tree networks for minimal Elmore delay, in *Proc. IEEE Int. Symp. On Circuits and Systems,* pp. 865–868, May 1990.

[Vittal 1995] A. Vittal and M. Marek-Sadowska, Power optimal buffered clock tree design, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 497–502, June 1995.

[Vittal 1996] A. Vittal, H. Ha, F. Brewer, and M. Marek-Sadowska, Clock skew optimization for ground bounce control, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 395–399, November 1996.

[Vuillod 1996] P. Vuillod, L. Benini, A. Bogliolo, and G. DeMicheli, Clock-skew optimization for peak current reduction, in *Proc. Int. Symp. on Low Power Electronics and Design,* pp. 265–270, August 1996.

[Wang 2005] K. Wang, Y. Ran, H. Jiang, and M. Marek-Sadowska, General skew constrained clock network sizing based on sequential linear programming, *IEEE Trans. on Computer-Aided Design*, 24(5), pp. 773–782, May 2005.

[Xi 1995] J. G. Xi and W. W.-M. Dai, Buffer insertion and sizing under process variations for low power clock distribution, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 491–496, June 1995.

[Xi 1996] J. G. Xi and W. W.-M. Dai, Useful-skew clock routing with gate sizing for low power design, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 383–388, June 1996.

[Yu 2007] Z. Yu, M. C. Papaefthymiou, and X. Liu, Skew spreading for peak current reduction, in *Proc. Great Lakes Symp. on VLSI,* pp. 461–464, March 2007.

R13.4 Power/Ground Network Design

[Bai 2000] G. Bai, S. Bobba, and I. N. Hajj, Simulation and optimization of the power distribution network in VLSI circuits, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 481–486, November 2000.

[Chen 1997] H. H. Chen and D. D. Ling, Power supply noise analysis methodology for deep-submicron VLSI chip design, in *Proc. ACM/IEEE Design Automation Conf,* pp. 638–643, June 1997.

[Chen 2001] T.-H. Chen and C. C.-P. Chen, Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative methods, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 559–562, June 2001.

[Chowdhury 1985] S. Chowdhury and M. Breuer, The construction of minimal area power and ground nets for VLSI circuits, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 794–797, June 1985.

[Director 1969] S. W. Director and R. A. Rohrer, The generalized adjoint network and network sensitivities, *IEEE Trans. on Circuit Theory*, 16(3), pp. 318–323, August 1969.

[Fu 2004] J. Fu, Z. Luo, X. Hong, Y. Cai, S. X.-D. Tan, and Z. Pan, A fast decoupling capacitor budgeting algorithm for robust on-chip power delivery, in *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Science*, E87-A(12), pp. 3273–3280, December 2004.

[Kozhaya 2001] J. N. Kozhaya, S. R. Nassif, and F. N. Najm, Multigrid-like technique for power grid analysis, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 480–487, November 2001.

[Kozhaya 2002] J. Kozhaya, S. Nassif, and F. Najm, A multigrid-like technique for power grid analysis, *IEEE Trans. on Computer-Aided Design*, 21(10), pp. 1148–1160, October 2002.

[Lin 2001] S. Lin and N. Chang, Challenges in power-ground integrity, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 651–654, November 2001.

[Mitsuhashi 1992] T. Mitsuhashi and E. Kuh, Power and ground network topology optimization for cell based VLSIs, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 524–529, June 1992.

[Oh 1998] J. Oh and M. Pedram, Multi-pad power/ground network design for uniform distribution of ground bounce, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 287–290, June 1998.

[Qian 2003] H. Qian, S. Nassif, and S. Sapatnekar, Random walks in a supply network, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 93–98, June 2003.

[Rothermel 1981] H.-J. Rothermel and D. A. Mlynski, Computation of power supply nets in VLSI layout, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 37–42, June 1981.

[Singh 2004] J. Singh and S. S. Sapatnekar, Topology optimization of structured power/ground networks, in *Proc. ACM Int. Symp. on Physical design,* pp. 116–123, April 2004.

[Singh 2005] J. Singh and S. S. Sapatnekar, A fast algorithm for power grid design, in *Proc. ACM Int. Symp. on Physical design,* pp. 70–77, April 2005.

[Su 2000] H. Su, K. Gala, and S. Sapatnekar, Fast analysis and optimization of power/ground networks, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 447–480, November 2000.

[Su 2003] H. Su, S. Sapatnekar, and S. Nassif, Optimal decoupling capacitor sizing and placement for standard-cell layout designs, *IEEE Trans. On Computer-Aided Design*, 22(4), pp. 428–436, April 2003.

[Syed 1982] Z. A. Syed and A. E. Carnal, Single layer routing of power and ground networks in integrated circuits, *J. of Digital Systems*, VI(1), pp. 53–63, Spring 1982.

[Tan 2003] X.-D. Tan, C.-J. Shi, and F. J.-C. Lee, Reliability-constrained area optimization of VLSI power/ground networks via sequence of linear programmings, *IEEE Trans. on Computer-Aided Design*, 22(12), pp. 1678–1684, December 2003.

[Xiong 1986] X.-M. Xiong and E. S. Kuh, The scan line approach to power and ground routing, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 6–9, November 1986.

[Zhao 2001] S. Zhao, K. Roy, and C.-K. Koh, Decoupling capacitance allocation for power supply noise suppression, in *Proc. ACM Int. Symp. on Physical design,* pp. 66–71, April 2001.

[Zhao 2002] M. Zhao, R. Panda, S. Sapatnekar, and D. Blaauw, Hierarchical analysis of power distribution networks, *IEEE Trans. on Computer-Aided Design*, 21(2), pp. 159–168, February 2002.

[Zhao 2004] M. Zhao, Y. Fu, V. Zolotov, S. Sundareswaran, and R. Panda, Optimal placement of power supply pads and pins, in *Proc. ACM/IEEE Design Automation Conf.,* pp. 165–170, June 2004.

[Zhong 2005] Y. Zhong and M. D. F. Wong, Fast algorithms for IR drop analysis in large power grid, in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design,* pp. 351–357, November 2005.